Sped. in Abb. Postale Gr. IV/70%

Anno XVIII - Numero 2 - Marzo-Aprile 1990 - L. 8.500

DESKTOP VIDEO
Otto sprite danzano per voi

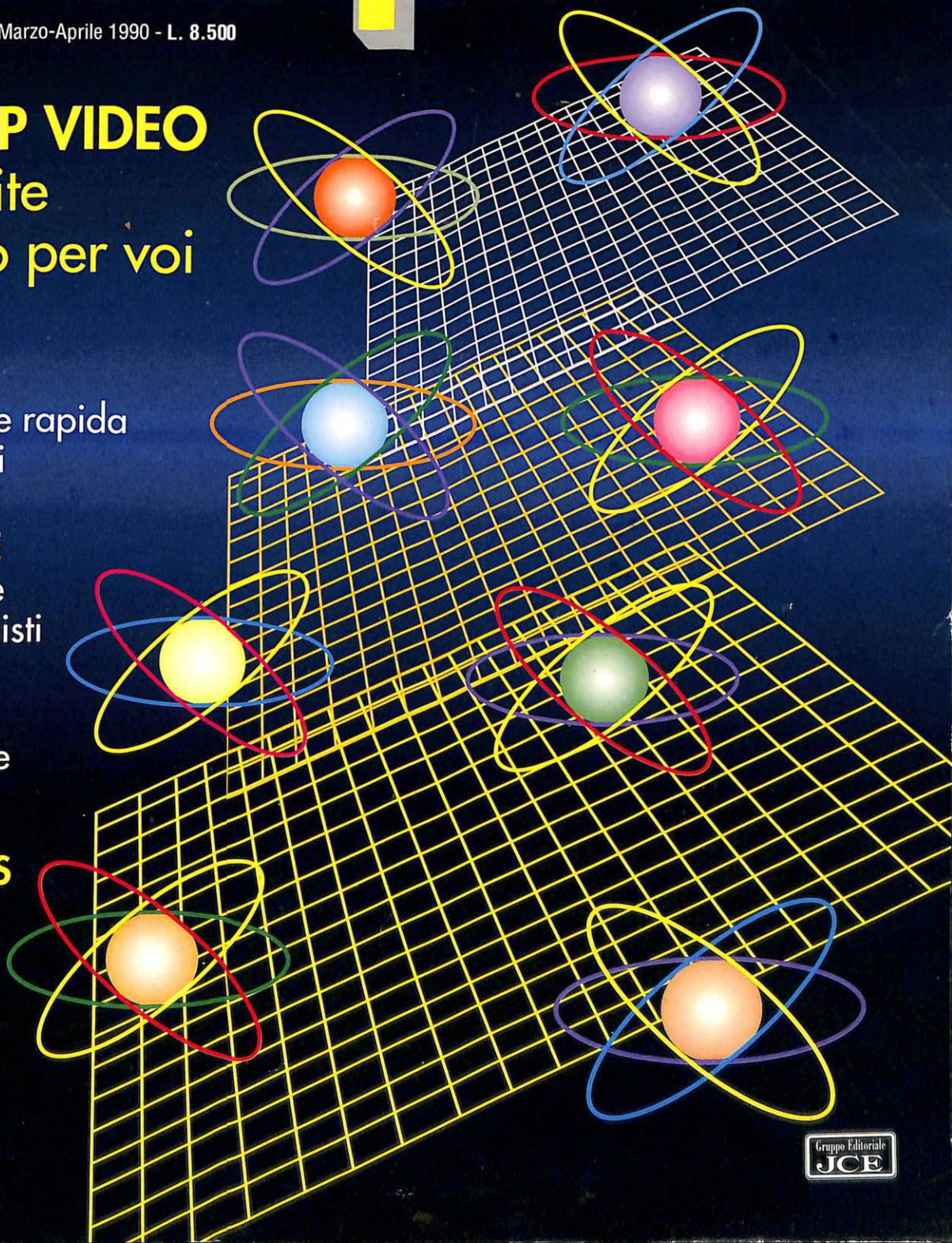
UTILITY
Compressione rapida di programmi

ESPANSIONE
Programmare da professionisti

ASSEMBLER
Dritto al cuore del sistema

TIPS & TRICKS
Vite infinite nei tuoi videogame

GIOCO
Eroica per amore

TASSA PAGATA PER CAMPIONE ALLEGATO

Tutto COMMODORE

La rivista per C64 e AMIGA

Anno III - Numero 31 - Marzo 1990 - L. 13.000

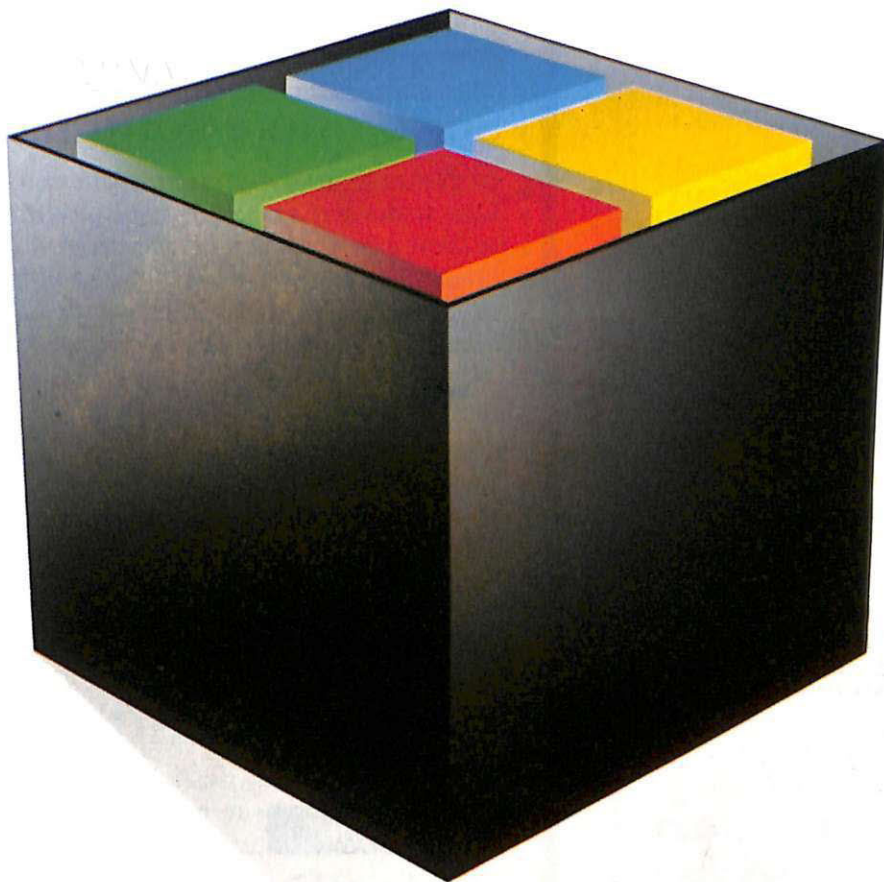
Sul disco: Desktop video col C64

- Introduzioni autoload
- Rastering avanzato
- 8 colonne sonore
- Effetto metallo sui testi
- Testi animati
- Animazioni e font personalizzabili

NEWS

Novità sul mercato italiano

IN PROVA

Video Printer stampa i segnali video di Amiga e C64

è in edicola

Direttore responsabile
Paolo Romani

Direttore Editoriale
Area Informatica
Marinella Zetti

Caporedattore
Fernando Zanini

Segretaria di redazione
Alessandra Marini

Art director
Sergio Sironi

Impaginazione elettronica
Adriano Barcella

Responsabile grafico
Desktop Publishing
Adelio Barcella

Collaboratori
Paolo Gussoni, Giorgio Caironi

Revisione testi
Antonella Cibelli, Flavia Ferro

RadioELETTRONICA&COMPUTER
Rivista bimestrale, una copia L. 8.500, numeri arretrati lire 13.000 cadauno.
Pubblicazione bimestrale registrata presso il Tribunale di Monza n. 679 del 28/11/88.

Fotolito: Bassoli - Milano.

Stampa: GEMM Grafica Srl, Paderno Dugnano (Mi).

Diffusione: Concessionario esclusivo per l'Italia A.&G. Marco Spa, via Fortezza 27 - 20126 Milano.
Spedizione in abb. post. gruppo III/70.

Abbonamenti: Annuale L. 64.000, estero L. 130.000.

Gruppo Editoriale JCE Srl
Sede legale, Direzione, Redazione,Amministrazione
via Ferri, 6, 20092 Cinisello Balsamo (Mi)
Tel. 02/66025.1 -Telex 352376 JCE MIL I -
Telefax 61.27.620 - 66.010.353

Direzione Amministrativa: Walter Buzzavo

Direttore Commerciale: Giogio Pancotti

Pubblicità e Marketing
Gruppo Editoriale JCE - Divisione Pubblicità
via Ferri, 6 - 20092 Cinisello Balsamo (Mi)
Tel. 02/66025.1

Responsabile Marketing: Daniela Morandi

Concessionario eslusivo per Roma, Lazio e centro sud:
UNION MEDIA Srl - via C. Fracassini, 18
00198 Roma - Tel. 06/3215434 (13 linee R.A.)
Telex 630206 UNION I - Telefax 06/3215678

Abbonamenti: Le richieste di informazioni sugli abbonamenti in corso si ricevono per telefono tutti i giorni lavorativi dalle ore 9 alle 12. Tel. 02/66025311- 66025338

I versamenti vanno indirizzati a:
Gruppo Editoriale JCE Srl, via Ferri, 6 20092 Cinisello Balsamo (Mi), mediante l'emissione di assegno circolare, cartolina vaglia o utilizzando il c.c.p. n. 351205. Per i cambi di indirizzo allegare alla comunicazione l'importo di L. 3.000, anche in francobolli e indicare insieme al nuovo anche il vecchio indirizzo.

SOMMARIO

N° 2
Marzo / Aprile 1990

Caricate così i programmi della cassetta allegata

*Riavvolgete il nastro e digitate **LOAD** seguito da **RETURN** sulla tastiera del C64 e **PLAY** sul registratore. Verrà caricato il programma di presentazione con il menù dei programmi. Digitate **RUN** seguito dalla pressione del tasto **RETURN**. Terminata la presentazione, per caricare uno qualsiasi dei programmi è sufficiente digitare: **LOAD "NOME PROGRAMMA"** seguito dalla pressione del tasto **RETURN**..*

LEGENDA

La difficolta di ogni programma di cui si parla all'interno della rivista è data dall'intensità di colore dei simboli Drive e Tape:

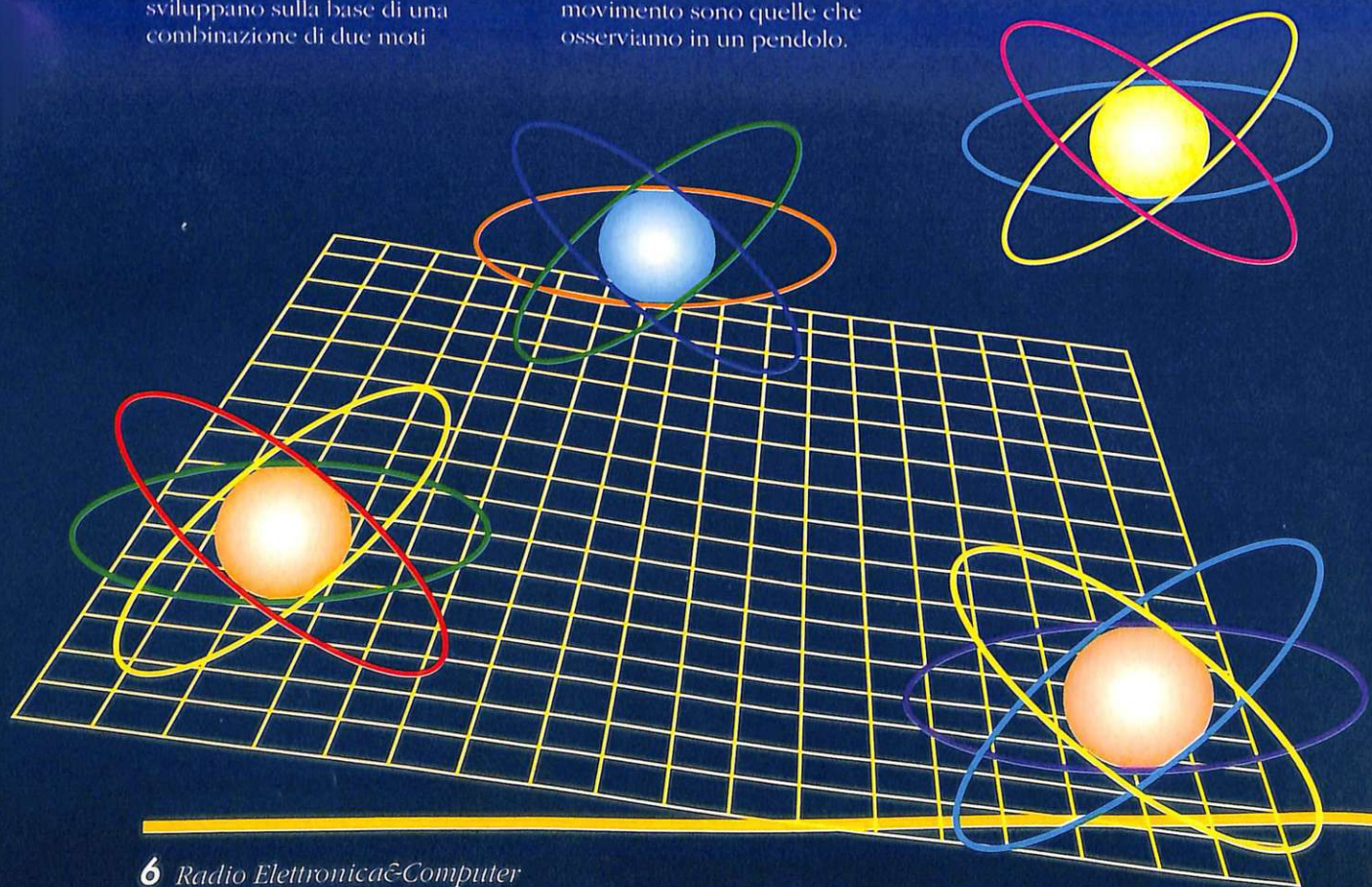
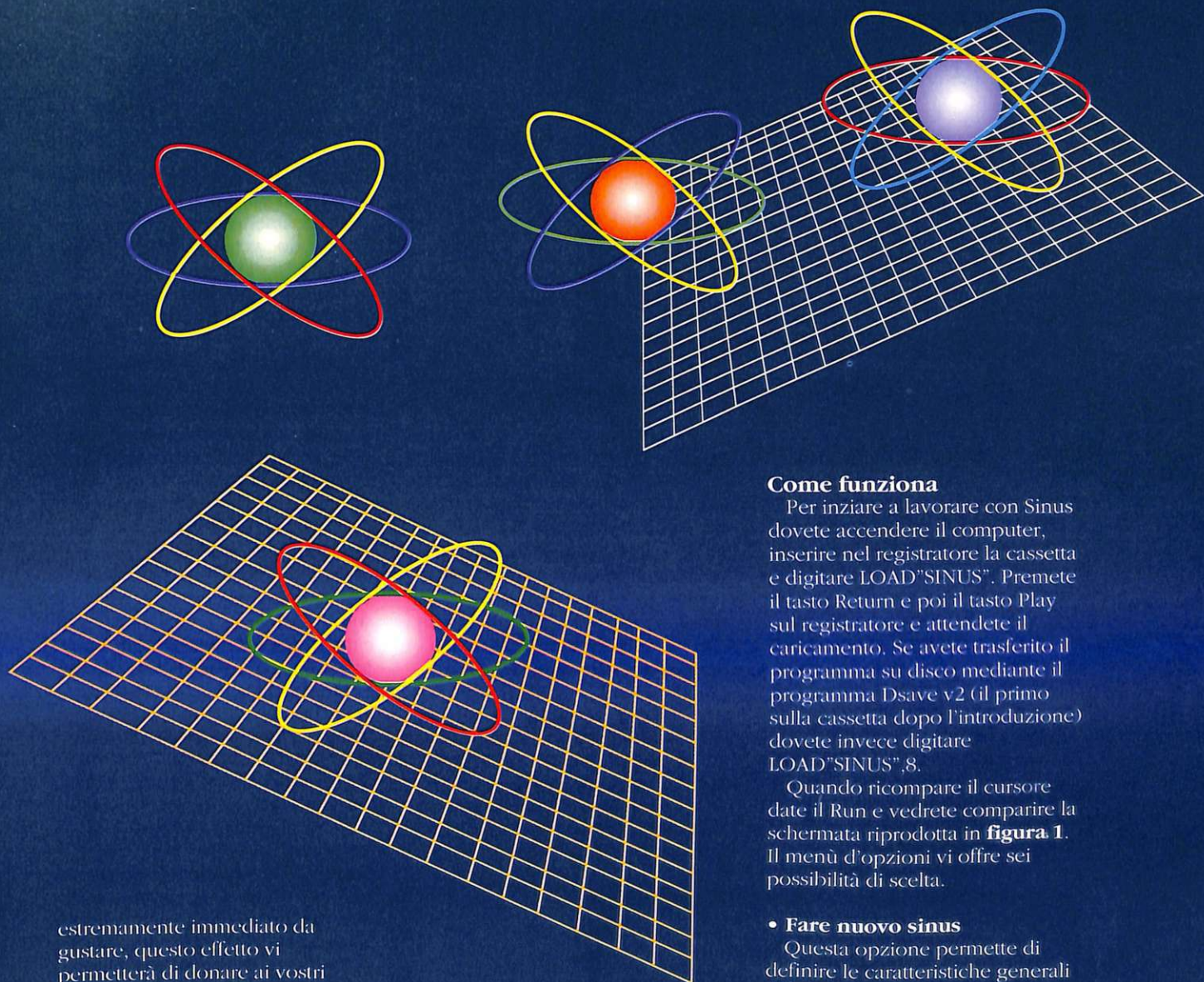
Testata di tiratura non superiore alle 15.000 copie (specializzata) e pertanto esente da certificazione obbligatoria, secondo regolamento CSST.

CSST — Associato al Consorzio Stampa Specializzata Tecnica

USPI — Mensile associato all'USPI Unione Stampa Periodica italiana

Sono otto e sono pazzi!

L'effetto speciale di questo numero è a tutto schermo! Otto sprite impazziti (a regola d'arte) si inseguono e si incrociano sullo schermo creando effetti ottici incredibili! Personalizzabilità e semplicità d'uso fanno di questo programma un vero gioiello per i collezionisti di effetti speciali e programmatori d'ogni tipo

Il programma Sinus è, come tutti i programmi di questa rubrica, un creatore di effetti speciali. L'effetto consiste nell'evoluzione di una serie di sprite. Le evoluzioni si sviluppano sulla base di una combinazione di due moti armonici. Un moto armonico non è altro che un particolare movimento di andata e ritorno lungo un segmento, da un estremo all'altro. Le caratteristiche del movimento sono quelle che osserviamo in un pendolo. L'aggettivo armonico definisce bene l'effetto visivo che tale movimento produce. Molto difficile da spiegare ma

estremamente immediato da gustare, questo effetto vi permetterà di donare ai vostri programmi un aspetto decisamente curioso e interessante. Una delle opzioni del Sinus consente di salvare un file programma di soli cinque blocchi che, opportunamente inizializzato e attivato, visualizza in interrupt l'effetto da voi creato. Effetto in interrupt significa che il vostro programma può girare come se il computer non stesse eseguendo le routine dell'effetto Sinus che quindi non vi crea alcun problema di programmazione. I programmi per creare effetti Sinus (che d'ora in poi chiameremo semplicemente Sinus) vanno fortissimi su personal computer come Amiga, ma il nostro buon vecchio C64 sa rimanere al passo con i tempi emulando egregiamente le incredibili prodezze grafiche del fratellone.

Come funziona

Per inziare a lavorare con Sinus dovete accendere il computer, inserire nel registratore la cassetta e digitare LOAD"SINUS". Premete il tasto Return e poi il tasto Play sul registratore e attendete il caricamento. Se avete trasferito il programma su disco mediante il programma Dsave v2 (il primo sulla cassetta dopo l'introduzione) dovete invece digitare LOAD"SINUS",8.

Quando ricompare il cursore date il Run e vedrete comparire la schermata riprodotta in **figura 1**. Il menù d'opzioni vi offre sei possibilità di scelta.

• Fare nuovo sinus

Questa opzione permette di definire le caratteristiche generali del sinus.

La prima caratteristica che vi è data di definire è il campo (**figura 2**). Per campo intendiamo l'area di schermo su cui volete si svolga il movimento. Inserite quindi un joystick in porta 2 e spostate i quattro limiti del campo come desiderate. Tenete presente che i limiti del campo possono essere mossi in due direzioni ciascuno con la combinazione del tasto Fire e la posizione, corrispondente al

Figura 1. La schermata principale del programma Sinus

limite di campo da muovere, sul joystick. Per esempio, per muovere a sinistra il limite sinistro è sufficiente spostare a sinistra la leva del joystick, mentre per muoverlo a destra dovete tenere premuto il tasto Fire compiendo la stessa azione. Provare è il modo migliore per capire. Nella parte superiore del video vedete degli indicatori che vi segnalano costantemente la posizione dei limiti. Quando avete terminato passate alla successiva sezione del programma con il tasto F1.

La seconda caratteristica su cui potete operare è la velocità dei due moti armonici che compongono il sinus. I due moti armonici si sviluppano orizzontalmente (moto X) e verticalmente (moto Y). Ciascun moto, benché non abbia mai una velocità costante, ha una velocità media che voi potete definire. Inoltre il programma permette di definire se questa velocità media possa variare durante la visualizzazione del sinus. Proprio a questa particolarità si riferisce la richiesta successiva alla definizione del campo. In sostanza vi è data la possibilità di decidere se la velocità media del moto possa variare alternativamente fra due da voi definite. Rispondendo affermativamente alla richiesta (Y) dovete inserire i due valori della velocità secondo il criterio visualizzato nella parte superiore dello schermo. Rispondendo negativamente potrete inserire solo un valore che decreterà costante la velocità media del moto relativo alla coordinata su cui state operando.

Quando avete terminato le operazioni di inserimento il computer calcolerà la curva in un minuto circa.

• Cambiare questo sinus

Questa funzione viene attivata automaticamente quando il programma ha terminato il calcolo della curva. Se la curva è già stata calcolata potete accedere a questa sezione del programma anche dal menù principale (tasto 2).

Il sinus è generato dai due moti armonici e dalle variazioni delle velocità medie di questi, ma non solo. Esiste, infatti, una caratteristica secondaria ma non irrilevante, che voi potete controllare. Come accennato all'inizio dell'articolo, il sinus gestisce fino a otto sprite. Ciascuno di essi inizia il percorso sulla curva con un leggero ritardo rispetto al precedente creando così un simpatico effetto a inseguimento. Il distacco fra uno sprite e l'altro costituisce l'oggetto delle richieste di input di questa sezione del programma. Il primo dato che vi viene richiesto è la distanza sull'asse x. In pratica questo dato specifica il ritardo nell'effettuare il percorso armonico sull'asse x per ciascun sprite. Per avere maggiori informazioni su questo argomento leggete il paragrafo Fare Pratica al termine di questo articolo.

Il secondo dato che viene richiesto riguarda la distanza sull'asse y fra gli sprite.

La questione è analoga alla precedente. Il terzo dato riguarda la distanza fra coordinate x e y, o

Figura 2. Impostazione dei limiti del campo

meglio il rapporto di sincronia fra i due moti armonici. Leggete l'ultimo paragrafo per maggiori informazioni.

L'ultimo dato in input riguarda il numero di sprite che si vuole prendano parte allo spettacolo. Sebbene uno sprite soltanto annulli gli effetti a inseguimento, è una scelta che non va esclusa a priori. Infatti, oltre a consentire una miglior analisi del moto creato, rimane comunque un effetto simpatico e incisivo per decorare i vostri programmi.

Figura 3. Menù dei colori

• Menù colori

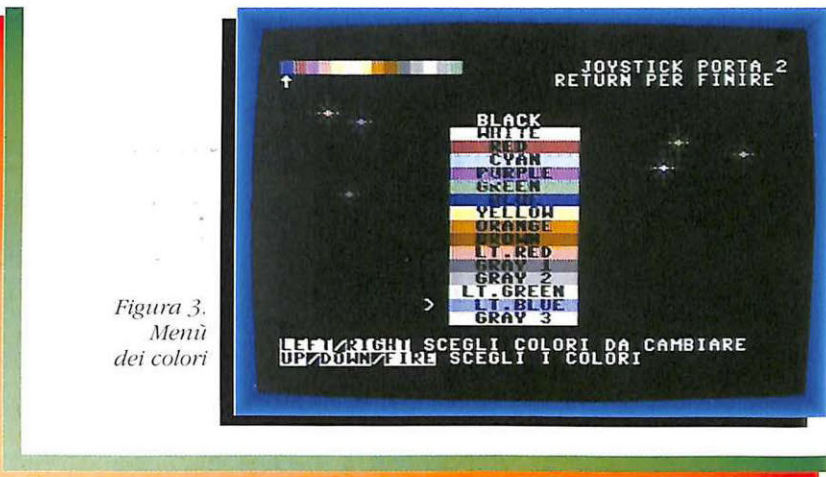
La pressione del tasto 3 dal menù principale introduce al menù dei colori (**figura 3**). Vediamo le cinque opzioni che lo compongono.

- Cambia colori

Il tasto A visualizza la schermata che vedete riprodotta in **figura 4**. Il joystick in porta 2 vi torna nuovamente utile in questa fase. Nella parte alta dello schermo c'è un indicatore che visualizza lo spettro dei colori. Una freccina, appena sotto a questo indicatore, indica su quale banda dello spettro state operando. La freccia può essere mossa spostando orizzontalmente la leva del joystick.

Nella parte centrale dello schermo avete la gamma di colori del Commodore 64. La freccia sulla sinistra può essere spostata muovendo verticalmente la leva del joystick. Premendo il tasto Fire assegnate il colore indicato al centro del video alla banda indicata in alto a sinistra. La funzione della banda colorata verrà illustrata più approfonditamente nei prossimi paragrafi.

- Velocità colori

Gli sprite in movimento possono cambiare colore in continuazione, secondo due modalità che vedremo in seguito. La frequenza con cui si verificano questi cambiamenti è il parametro che vi è dato di regolare in questa sezione del programma. Alla richiesta di input dovete inserire un valore compreso fra 1 (velocità massima) e 255 (velocità minima).

- Colori on/off

Abbiamo detto prima che i colori possono cambiare oppure no. La definizione di questa opportunità viene effettuata tramite questa funzione. Alla richiesta di input rispondete con Y (yes) per decretare il cambiamento continuo dei colori, oppure N per ottenere l'effetto opposto.

- Flash/Burst

Se l'effetto di cambiamento dei colori è stato attivato, è possibile stabilire due modalità di comportamento per questo effetto. L'effetto in modo Flash considera tutti gli sprite di un colore unico che cambia ciclando secondo l'ordine specificato nello spettro (quello menzionato a proposito della funzione Cambia Colori) e a una certa velocità (secondo quanto definito con la funzione Velocità Colori).

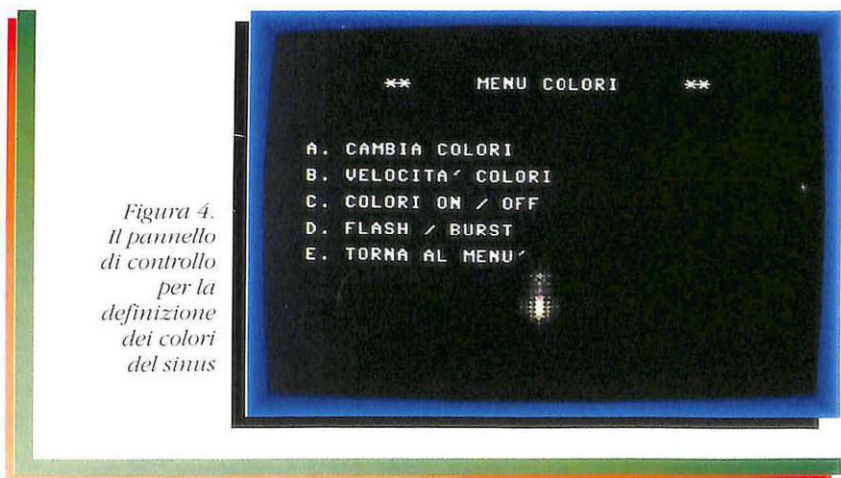



Figura 4. Il pannello di controllo per la definizione dei colori del sinus

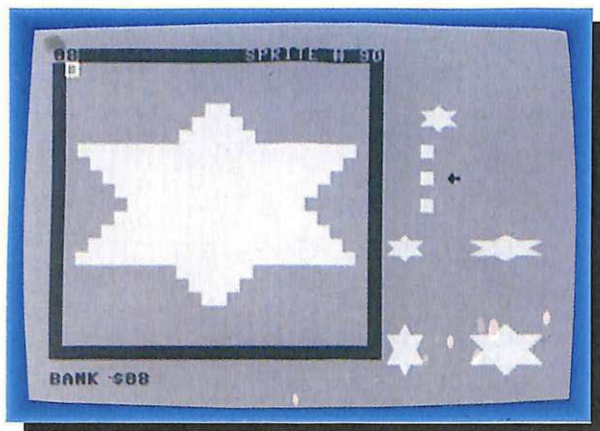



Figura 5. Un esempio di sprite realizzato con il programma Editor comparso sul numero scorso

L'effetto di tipo Burst, invece, cicla i colori partendo dal primo della catena. Il procedimento, per ciascun sprite della catena, rimane sfasato di un colore rispetto allo sprite adiacente. L'effetto che ne scaturisce è decisamente efficace.

L'ultima opzione del menù dei colori permette di tornare al menù principale.

• Save Sinus

Il salvataggio del sinus è la funzione clou del programma. Salvare il sinus, infatti, non significa solo poterlo archiviare, ricaricare e rielaborare in seguito, ma anche utilizzarlo nei propri programmi con grande arricchimento estetico degli stessi. Attivata la funzione di saving, il computer chiede se si desidera salvare l'intero sinus (tasto I) o solo i dati della curva calcolata (tasto D). La prima scelta salva sul disco un file di cinque blocchi. Questo file (se caricato con LOAD"nome",8,1) risiede in memoria dalla locazione $7000 (28672) alla locazione $73FF (29695). Nell'ultimo paragrafo di questo articolo vedremo come utilizzare questo file nei vostri programmi.

La seconda scelta salva su disco solo i dati che definiscono la curva del sinus calcolato. Sempre nell'ultimo paragrafo vedremo come utilizzare anche questa funzione.

• Load Sinus

Esistendo la funzione Save non poteva certo mancare la funzione Load. Premendo il tasto 5 compare la parola "load" seguita dal simbolo $ (directory). Premendo direttamente il tasto Return verrà visualizzata la directory del disco. Viceversa, specificando il nome del file ne segue il caricamento. Il nome del file può essere un sinus salvato precedentemente o uno sprite, purché questo sia già rilocato nella posizione opportuna.

• Vedere

L'ultima opzione del menù è stata pensata per consentire di vedere l'effetto sinus senza i testi di sfondo. Per tornare al menù occorre premere la barra spazio e il tasto Return.

Fare pratica

In questo paragrafo cercheremo di chiarire bene il funzionamento della routine di generazione del sinus e dare al lettore i consigli necessari per sfruttare al meglio il sinus nei propri programmi.

Appena lanciato il programma compare la schermata di **figura 1**. Premete il tasto 3 ed entrate nel menù dei colori. Premete il tasto B e digitate 0 (è un trucchetto per rendere lentissimo il cambiamento dei colori). A questo punto premete il tasto D e poi il tasto F. Premete il tasto C e, quando gli sprite hanno assunto un colore che ritenete ben visibile premete N. A questo punto gli sprite manterranno il colore selezionato per tutto l'esperimento. Premete il

tasto E per tornare al menù principale. Ora premete il tasto 1 e, con l'aiuto del joystick in porta 2 fissate i limiti del campo (definite un'area più piccola dell'intero schermo) e premete, al termine, F1. Impostate una velocità unica per entrambi i moti X e Y. Per il moto X fissate una velocità di quattro, mentre per il moto Y una velocità di zero. Quando il programma ha calcolato la curva, vi viene chiesto di specificare i tre valori che quantificano la distanza e la sincronizzazione degli sprite. Impostate i valori a zero e, all'ultima richiesta sul numero degli sprite da utilizzare, specificate 1. Quello che vedete sullo schermo è il moto X. Lo abbiamo isolato per semplificare la comprensione del meccanismo che genera il sinus. Ora immaginate che a questo moto orizzontale venga aggiunto, per lo stesso sprite, un moto dello stesso tipo, ma verticale. Riuscite a immaginare come si muoverà lo sprite? Attivate nuovamente la prima opzione del menù principale e fissate a quattro anche la velocità del moto Y (verticale). Questo è il concetto fondamentale che governa il comportamento di un sinus, ma la pratica vi aiuterà molto di più a capire come funziona tutto il sistema. Ricordate solo di non esagerare con i valori delle distanze fra sprite perché questo crea molta confusione sul video. Naturalmente, quando avrete ben chiari gli effetti di ogni funzione, anche l'effetto confusione sarà un potente strumento per rendere spettacolari vostri sinus.

Il file generato dal programma è facilissimo da usare: resettate il computer e caricate il file con LOAD"nome",8,1. Digitate:

POKE 53269,255

per attivare gli sprite e premete Return. Dopo digitate:

SYS 28672

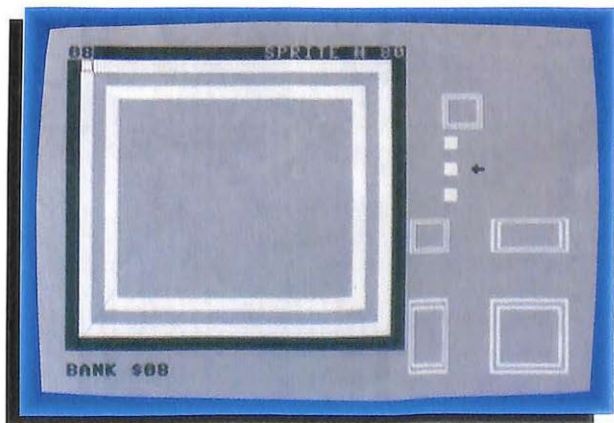
per attivare l'interrupt. Se avete resettato il Commodore 64 spegnendolo vedete il vostro sinus eseguito da una serie di sprite di forma casuale. Infatti dovreste aver caricato uno sprite prima, ma affronteremo più avanti questa questione.

La cosa importante è notare che il cursore (e il computer) è ancora a vostra disposizione per caricare programmi e programmare. Premendo Run/Stop e Restore arresterete l'interrupt che è comunque riattivabile secondo la procedura vista prima. In un vostro programma, tutto quello che dovete fare è caricare uno sprite con un normalissimo LOAD, attivare gli sprite con la POKE vista sopra, attivare l'interrupt con la SYS. A proposito dello sprite osservate le **figure 5**, **6**, **7**, per avere qualche idea. Lo sprite può essere creato in qualsiasi modo, l'importante è che, caricato, vada a rilocarsi in $2400 (9216).

Il programma Editor comparso sul numero scorso di *Radio Elettronica & Computer* è l'ideale (sprite $90).

Studio Bitplane

Figura 6. Lo sprite di default del programma Sinus

Figura 7. Un'altra idea per uno sprite: questa forma genera un'interessante effetto tunnel

Ecco un gioco dalla trama appassionante e dalla grafica eccellente. Dovrete vivere una tragica storia d'amore e lottare contro il destino per trasformarla in una fiaba d'amore e felicità. Divertimento assicurato

Eroica per amore

Circa un anno fa il mondo dei videogiochi ha conosciuto una novità destinata a far parlare di sé per molto tempo: dalla ReadySoft era giunta in Italia la versione per Amiga di Dragon's Lair, meglio noto come "Il videogioco laser". La versione coin-op di Dragon's Lair era diventata famosa perché era stato il primo tentativo di realizzare un videogioco basato su un cartone animato, registrato su un videodisco. Come tutti ricorderete, per giocare con Dragon's Lair è necessario seguire l'eroe nella sua avventura guidandolo correttamente con precisi colpi di joystick per fargli oltrepassare tutti gli ostacoli che gli sbarrano la strada. A ogni ostacolo, se si agisce correttamente, il cartone animato procede regolarmente, viceversa se si sbaglia qualche mossa si assiste al fallimento della missione.

Sembrava impossibile riprodurre Dragon's Lair su un personal computer comune e invece i tecnici della ReadySoft sono riusciti a realizzare l'animazione suddividendo l'avventura in tante parti e salvando i diversi fotogrammi del cartone animato

sotto forma di schermate di Amiga su ben sei floppy disk, per un totale di oltre 5 Mb di memoria.

La ben nota velocità di Amiga nel gestire le pagine grafiche ha fatto il resto e così è nata la spettacolare versione di Dragon's Lair per Amiga. Tra parentesi quel programma diventò rapidamente noto anche tra i pirati del software, poiché la ReadySoft studiò per questo eccezionale prodotto un sistema di protezione anti-copia davvero efficace. È vero che oggi circola una copia piratata di Dragon's Lair, ma essa richiede ben otto floppy disk, inoltre ha visto la luce solo alcuni mesi dopo l'uscita dell'originale. In linea di principio lo stesso procedimento può essere utilizzato anche con il C64, anche se naturalmente non è possibile ottenere un'animazione altrettanto buona poiché il 6510 non ha certo la stessa capacità di gestire blocchi di memoria del blitter di Amiga, inoltre disponendo di soli 64 kb di Ram non è possibile mantenere in memoria molte schermate contemporaneamente.

Comunque sia l'idea ci è piaciuta e abbiamo deciso di creare un videogioco originale utilizzando la tecnica di Dragon's Lair. Per fare questo, dapprima abbiamo inventato una storia sulla quale basare l'avventura, poi abbiamo disegnato tutte le scene della missione e infine abbiamo creato un programma che gestisce l'animazione e il gioco.

Ovviamente il risultato non è paragonabile a Dragon's Lair, né potrebbe esserlo poiché la produzione di un videogioco di grande successo commerciale non è basata sulla fantasia e la creatività di un solo programmatore, bensì è il risultato di un organizzatissimo lavoro di squadra cui partecipano esperti di programmazione, di grafica, di animazione e di effeti sonori. Resta comunque fondamentale il

contributo del geniaccio che inventa la struttura e la dinamica del gioco. Il nostro programma non ha tante pretese, essendo un lavoro dal sapore perlopiù artigianale, tuttavia si tratta di un esperimento interessante che non mancherà di divertirvi e magari di suggerirvi qualche idea per un vostro videogioco. Iniziamo a vedere la trama della storia su cui si basa il gioco.

La storia è molto triste: Inge, una bellissima ragazza bionda, è follemente innamorata di Hans, il quale a sua volta è legato a lei da un profondo affetto. Essi costituiscono una coppia bellissima e molto affiatata, pertanto sembra impossibile che qualcosa li possa dividere. Ma com'è noto, anche se solitamente fingiamo di dimenticarcene, polvere siamo e polvere torneremo a essere e così Hans muore in un incidente aereo.

Il dolore di Inge è immenso, la bellissima ragazza è inconsolabile. Inge non riesce ad arrendersi all'idea di non vedere più il suo amato Hans, ed è sull'orlo della pazzia. Non volendo lasciare nulla di intentato, Inge decide di rivolgersi a una chiromante, nella speranza che la magia possa aiutarla a rivedere Hans.

La vecchia chiromante si commuove per la storia dell'amore sfortunato di Inge e Hans e così decide di aiutarla a riavere l'amato compagno. L'unico modo per riportare in vita Hans, spiega la chiromante alla ragazza, è il woodoo, una pratica magica che riporta in vita i morti. Affidarsi a queste pratiche magiche può essere molto pericoloso, ma Inge decide di affidarsi all'aiuto della vecchia chiromante, pronta ad affrontare qualsiasi prova: meglio la morte della vita senza Hans.

La vecchia accompagna Inge in un bosco, dove l' abbandona al suo destino. Per riuscire nell'impresa ha bisogno dell'aiuto di un'anima pia che l'accompagni in una difficile strada sotterranea che porta al diavolo. Solo uccidendo il diavolo Inge potrà riavere il suo amato Hans, che risorgerà ai piedi del cadavere del malefico.

Inutile aggiungere che l'anima pia in questione siete voi e dalla

Tavola 1.

I file che devono essere trasferiti dalla cassetta al disco dopo l'uso di Genesi.

0.AG	*25*	*B*
1.AG	*130*	*B*
2.AG	*25*	*B*
3.AG	*130*	*B*
4.AG	*25*	*B*
5.AG	*118*	*B*
7.AG	*17*	*B*
8.AG	*25*	*B*
9.AG	*93*	*B*
C.AG	*2*	*B*

vostra abilità di decidere in fretta i movimenti di Inge dipende la vita della ragazza, nonché ovviamente il successo della missione.

Al lavoro dunque, una difficile avventura negli inferi vi aspetta.

I file che costituiscono il programma sono sulla cassetta.

In **tavola 1** c'è l'elenco dei moduli che dovete trasferire su disco. Un caso particolare è costituito dal programma Genesi, il primo dei file del gioco, anch'esso sulla cassetta. Ecco come dovete operare per costruire il disco master del gioco. Procuratevi un disco vuoto e formattato (basta una sola facciata). Caricate e lanciate dalla cassetta il programma Genesi (che non è necessario trasferire su disco). Seguendo le elementari istruzioni otterrete un disco (il master che avete preparato) su cui si trovano quattro file: BOOT, T.AG, A.AG e 6.AG. A questo punto dovete ricorrere al programma Dsave v2 (anch'esso sulla casetta) e trasferire i programmi elencati in **tavola 1**, sul disco master.

Per caricare il gioco bisogna utilizzare l'istruzione LOAD"*",8,1 oppure LOAD"C*",8,1+RUN.

Dopo il caricamento dei primi file compare una schermata di presentazione. Per iniziare il gioco si deve inserire il joystick in porta 2 e si deve premere Fire. Segue un rapido riassunto della vicenda di Inge, corredato di alcune belle illustrazioni. Per far avanzare le pagine si deve tenere premuto il tasto Fire. Durante lo svolgimento del gioco il floppy disk deve essere sempre lasciato nel drive perché il programma è lungo circa 150 kb, pertanto richiede il periodico accesso al disco.

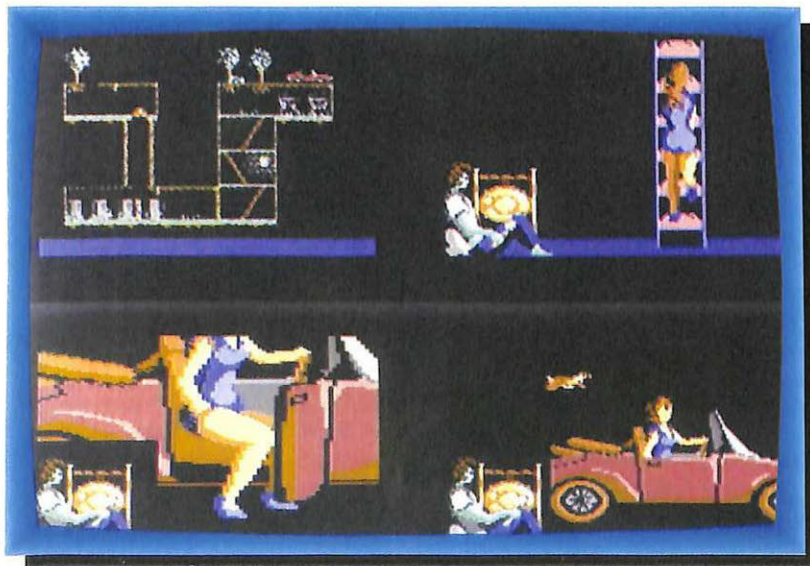
È molto utile un velocizzatore parallelo tipo speed-dos per accelerare il caricamento dei vari file. Terminata la presentazione inizia l'avventura vera e propria, con Inge all'ingresso del regno degli inferi e un velenoso serpente che l'attacca. La vostra prima mossa deve allontanare Inge dal serpente: se superate questa prima difficoltà vi trovate davanti a un'altra situazione difficile e così via finché non raggiungete la vostra meta, oppure non perdete tutte le vite a vostra disposizione. Ogni situazione richiede un movimento particolare del joystick che dipende dalla particolare situazione: se agite correttamente la scena prosegue come un cartone animato, mentre se sbagliate o se ritardate troppo vi tocca assistere alla fine prematura di Inge. In certe occasione potete controllare direttamente i movimenti di Inge con il joystick.

Se riuscite a percorrere tutta la strada che vi separa dal temibile diavolo dovete affrontarlo. Per aver ragione del vostro nemico dovete individuarne il punto debole e dovete colpirlo centrandolo con il mirino.

Se portate a termine la missione potete assistere al ritorno di Hans ai piedi del diavolo.

A ogni tappa del viaggio corrisponde un certo punteggio: se superate quota 3.500 avete diritto a inserire il vostro nome nella classifica di Demon Lover.

Siamo sicuri che, pur essendo un prodotto "artigianale", Demon Lover riscuoterà un certo successo: non vi resta che prepararvi ad affrontare i misteri degli inferi.

Alberto Geneletti
Gianni Arioli

Basic più...

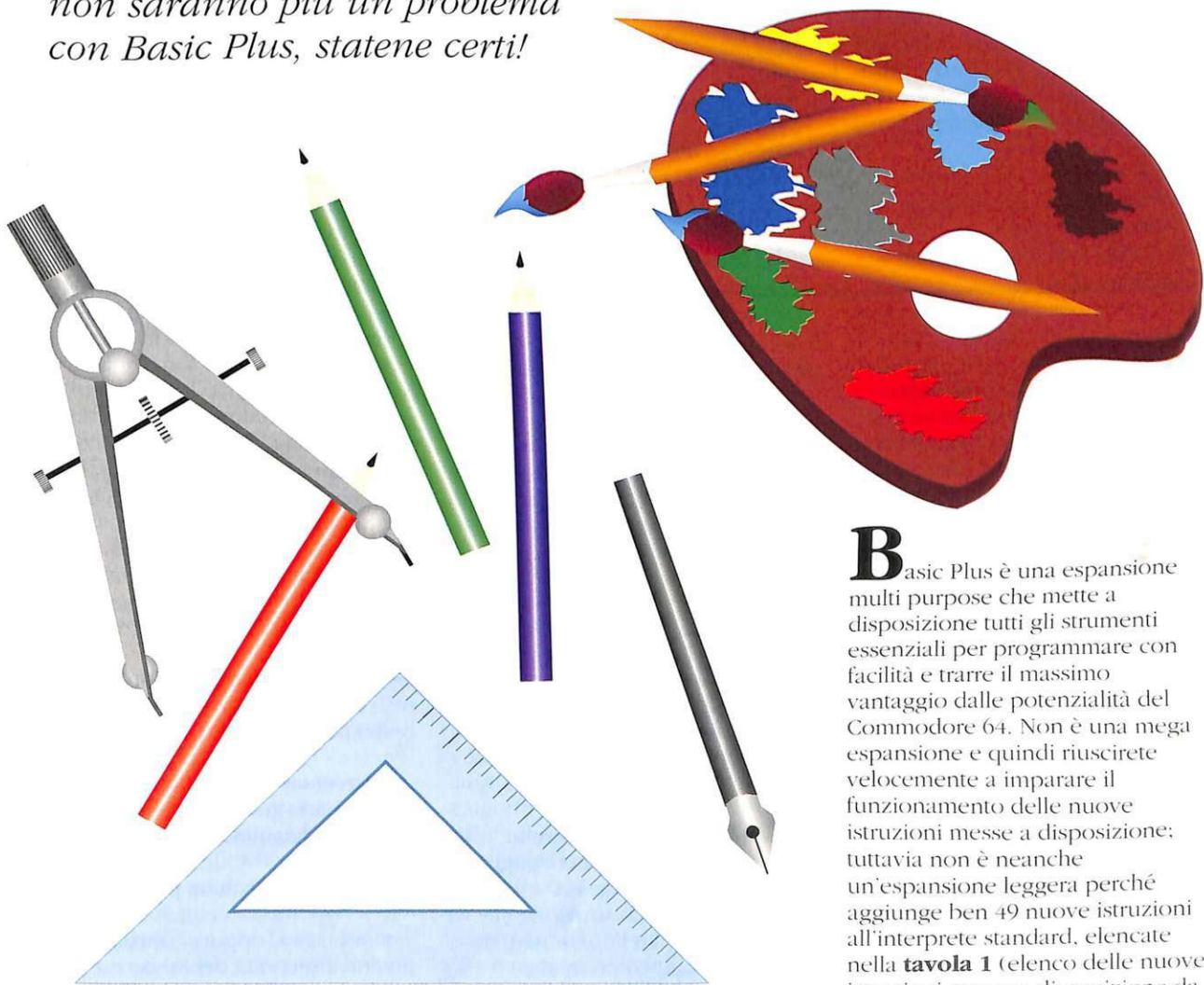
L'espansione di questo mese si chiama Basic Plus ed è una potentissima espansione tuttofare. Sprite, alta risoluzione e debug dei programmi non saranno più un problema con Basic Plus, statene certi!

Basic Plus è una espansione multi purpose che mette a disposizione tutti gli strumenti essenziali per programmare con facilità e trarre il massimo vantaggio dalle potenzialità del Commodore 64. Non è una mega espansione e quindi riuscirete velocemente a imparare il funzionamento delle nuove istruzioni messe a disposizione; tuttavia non è neanche un'espansione leggera perché aggiunge ben 49 nuove istruzioni all'interprete standard, elencate nella **tavola 1** (elenco delle nuove istruzioni messe a disposizione da Basic Plus). Come di consueto raccoglieremo le istruzioni in gruppi secondo l'ambito di applicazione.

Tavola 1.

Hires l, f	*Sprite sn, c*
Cls	*Volume n*
Graph	*Envelope v, a, d, s, r.*
Color l, f, x1, y1, x2, y2	*Wave v, w*
Mode n	*Play v, f, l oppure play v, "stringa-note", l.*
Inverse	*Renumber s*
Set x, y	*Delete i, f*
Line x1, y1, x2, y2	*Auto n*
Move x, y	*Find stinga*
Circle x, y, xr, yr	*Merge*
Fill x, y	*Save "nomefile", ndv, is, init, fine*
Text x, y, b, (sc$+)"testo"(, xf, yf)	*@ ndv, stringa-comando*
Box x1, y1, x2, y2	*Print@ l, c, "stringa"*
Frame x1, y1, x2, y2	*Restore ln*
Gload	*Doke loc, val*
Gsave	*Allclose*
Mobdef sn, b	*Inkey l, c, maxc, a$, r$(, fl)*
Mobsize sn, xe, ye, p	*Hardcopy n*
Mobset sn, x, y	*Case error goto*
Mobcolor sn, c	*Resume*
Mobmulti sn, c1, c2, c3	*Usr (n)*
Mobmove sn, x1, y1, x2, y2, v	

Grafica

• **Hires:** attiva l'alta risoluzione. La sintassi di questa istruzione è: Hires l, f dove il primo parametro rappresenta il colore di linea e il secondo il colore del fondo dello schermo. Questa istruzione cancella il contenuto della pagina grafica e attiva automaticamente il modo 0 di funzionamento delle istruzioni grafiche (vedere l'istruzione Mode). Il colore del bordo dello schermo è quello fissato in bassa risoluzione (modo testo). Hires può essere utilizzata sia in modo diretto sia in modo programma. Per ritornare in bassa risoluzione basta utilizzare l'istruzione Norm (descritta più avanti) oppure premere il tasto Run/Stop. Per esempio per attivare l'alta risoluzione utilizzando il bianco come colore di linea e il nero come colore di fondo basta scrivere:

hires 1, 0

• **Cls:** cancella la pagina grafica e non necessita di alcun parametro. Ovviamente questa istruzione va preferita alla precedente per cancellare la pagina grafica sia perchè leggendola nel listato si sa subito come si comporterà il programma e sia perchè non richiede di specificare i due parametri dei colori.

• **Graph:** attiva l'alta risoluzione senza cancellare la pagina grafica. Questa istruzione, che non necessita di alcun parametro, è quindi utilissima per sbirciare la pagina grafica senza alterarne il contenuto.

• **Color:** cambia il colore di linea e di fondo dell'intera pagina grafica oppure di una sua sottoparte. Questa istruzione è comodissima perché permette di usare tantissimi colori in alta risoluzione evitando la necessità di utilizzare il modo multicolor. L'istruzione può essere utilizzata in due diversi formati: color l, f oppure color l, f, x1, y1, x2, y2. Nel primo caso l'istruzione agisce sull'intera pagina grafica mentre nel secondo caso agisce solo sulla sezione specificata attraverso gli ultimi quattro parametri. In entrambi i casi i parametri l e f hanno lo stesso significato degli omonimi parametri dell'istruzione Hires. I parametri x1, y1, x2 e y2 rappresentano rispettivamente le coordinate, in caratteri (o locazioni video), dell'angolo superiore sinistro e inferiore destro del rettangolo di pagina grafica in cui cambiare i colori.

I parametri che iniziano con x possono assumere un valore qualsiasi nel range 0-39 mentre i due parametri che iniziano con y possono assumere un valore qualsiasi nel range 0-24. Per esempio, se vogliamo cambiare il colore di linea e di fondo delle prime due linee dello schermo in nero e blu rispettivamente basta scrivere:

color 0, 6, 0, 0, 39, 1

• **Mode:** fissa il modo di funzionamento delle istruzioni grafiche, cioè il modo in cui saranno trattati i punti cui si accede attraverso le istruzioni grafiche. La sintassi dell'istruzione è mode n dove n può assumere i seguenti valori:

n=0: i punti vengono settati
n=1: i punti vengono cancellati
n=2: i punti vengono invertiti

• **Inverse:** inverte il contenuto della pagina grafica, cioè pone in reverse l'immagine visualizzata in alta risoluzione. Questa istruzione non necessita di alcun parametro.

• **Set:** setta (oppure cancella o inverte a seconda del modo di funzionamento fissato con Mode) un punto della pagina grafica. La sintassi è: set x, y dove x e y rappresentano l'ascissa e l'ordinata del punto cui si vuole accedere.

Ecco un piccolo esempio d'uso:

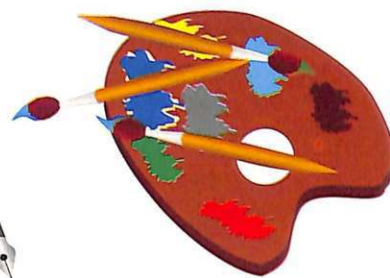
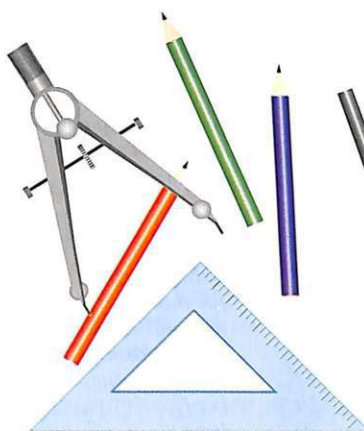
x=100: y=abs(100*sin(x)): set x,y

I due parametri che devono essere passati all'istruzione devono essere compresi rispettivamente nei range 0-319 e 0-199.

• Line: disegna una linea. La sintassi è line x1, y1, x2, y2 dove i quattro parametri rappresentano le coordinate dei due punti fra cui deve essere tracciata la linea. Per i parametri di questa istruzione valgono le stesse limitazioni viste per l'istruzione precedente (le ascisse devono essere comprese fra 0 e 319 e le ordinate fra 0 e 199).

• Move: è un'istruzione abbastanza insolita, serve infatti per trascinare in una determinata direzione l'ultimo punto trattato con Line o Set. Il punto trascinato setterà, cancellerà o invertirà i punti che tocca durante il movimento a seconda del modo attivo. La sintassi dell'istruzione è move x, y dove x e y rappresentano le coordinate in cui il punto in questione verrà spostato. Ecco un esempio d'uso:

line 1,1,30,40: move 100, 65.

• Circle: disegna un'ellisse nella pagina grafica. La sintassi dell'istruzione è circle x, y, xr, yr.
I primi due parametri rappresentano le coordinate del centro dell'ellisse mentre gli ultimi due parametri sono rispettivamente il semiasse orizzontale e verticale. Con questa istruzione si possono anche disegnare dei cerchi, basta fare in modo che xr e yr siano uguali. I primi due parametri devono essere compresi nei range 0-319 e 0-199 mentre gli ultimi due devono essere compresi, entrambi, nel range 0-128.

• Fill: è l'arcinota istruzione per riempire una regione chiusa della pagina grafica. La sintassi è fill x, y dove i due parametri rappresentano le coordinate di un punto qualsiasi interno alla regione che si vuole riempire.

• Text: questa utilissima istruzione permette di scrivere nella pagina grafica utilizzando caratteri di dimensioni variabili. La sintassi è text x, y, b, (sc$+)"testo"(, xf, yf). Vediamo il significato dei parametri. Innanzitutto va sottolineato che tutti i parametri che compaiono tra parentesi sono opzionali; tuttavia se specificate il parametro xf dovete specificare anche il parametro yf e viceversa. I primi due parametri rappresentano le coordinate del punto da cui si vuole iniziare a visualizzare la stringa. Per questi parametri valgono le usuali limitazioni sui valori che possono assumere. Il parametro b indica la distanza, in pixel, fra due caratteri successivi (il valore normale per questo parametro è 8). Il parametro sc$, che è opzionale, è una stringa di controllo che può assumere i seguenti valori:

sc$= chr$(133): visualizza la stringa da sinistra verso destra
sc$= chr$(134): visualizza la stringa da destra verso sinistra
sc$= chr$(135): visualizza la stringa dall'alto verso il basso
sc$= chr$(136): visualizza la stringa dal basso verso l'alto

Gli ultimi due parametri, opzionali, permettono di fissare la larghezza e l'altezza dei caratteri: rappresentano infatti il fattore di cui verranno espansi i caratteri nelle direzioni orizzontale e verticale rispetto alle dimensioni standard (8 x 8 pixel). Ecco un esempio d'uso di questa istruzione:

```
10 hires 0,1
20 fort=0 to 3: text 100, 100, 8,
chr$(133+t)+"123", 2, 3: next
30 goto 30
```

Lanciato questo breve demo, per fare ritorno al modo testo dovete premere il tasto Run/Stop.

• Box: traccia un rettangolo pieno nella pagina grafica.
La sintassi è box x1, y1, x2, y2 dove i primi due parametri rappresentano le coordinate dell'angolo superiore sinistro del rettangolo mentre gli ultimi due sono le coordinate dell'angolo inferiore destro.

• Frame: traccia un rettangolo nella pagina grafica. La sintassi di questa istruzione è identica a quella dell'istruzione precedente (la differenza fra le due istruzioni sta nel fatto che Frame disegna un

Tavola 2.

1	*too many files*	*16*	*out of memory*
2	*file open*	*17*	*undef'd statement*
3	*file not open*	*18*	*bad subscript*
4	*file not found*	*19*	*redim'd array*
5	*device not present*	*20*	*division by zero*
6	*not input file*	*21*	*illegal direct*
7	*not output file*	*22*	*type mismatch*
8	*missing filename*	*23*	*string too long*
9	*illegal device number*	*24*	*file data*
10	*next without for*	*25*	*formula too complex*
11	*syntax error*	*26*	*can't continue*
12	*return without gosub*	*27*	*undef'd function*
13	*out of data*	*28*	*verify*
14	*illegal quantity*	*29*	*load*
15	*overflow*		

rettangolo vuoto mentre Box traccia un rettangolo pieno).

• **Gload:** ha la stessa sintassi dell'istruzione Load del Basic standard e permette di caricare una pagina grafica precedentemente salvata con Gsave.

• **Gsave:** ha la stessa sintassi dell'istruzione Save del Basic standard e permette di salvare il contenuto della pagina gafica.

Sprite

• **Mobdef:** definisce uno sprite, cioè associa a uno sprite il blocco di 64 byte in memoria che ne contiene la definizione. La sintassi è Mobdef sn, b dove sn è il numero dello sprite, un intero fra zero e sette, mente b è il numero del blocco che ne contiene la definizione (il primo blocco parte dalla locazione 0).

• **Mobsize:** fissa le dimensioni orizzontale e verticale di uno sprite e la sua priorità rispetto ai caratteri. La sintassi è mobsize sn, xe, ye, p. Il primo parametro è il numero dello sprite. Gli altri due parametri successivi indicano se lo sprite deve essere espanso (parametro = 1) in direzione orizzontale e verticale oppure no (parametro = 0). Il parametro p fissa invece la priorità dello sprite rispetto ai caratteri: se p vale 0 allora lo sprite non ha la priorità sui caratteri mentre se vale 1 ce l'ha.

• **Mobset:** posiziona uno sprite. La sintassi è mobset sn, x, y. Il significato dei parametri dovrebbe essere facilmente intuibile: sn rappresenta il codice dello sprite mente x e y sono le coordinate in cui lo sprite deve essere posto. Il parameto x può esssere un intero qualsiasi compreso fra 0 e 512 (non avrete quindi più a che fare con il famigerato most significant bit!) mentre y può assumere un valore intero qualsiasi fra 0 e 255.

• **Mobcolor:** fissa il colore di uno sprite. La sintassi è mobcolor sn, c dove sn è il codice dello sprite e c il codice del colore (il colore che si fissa con questa istruzione è quello indipendente per ogni sprite).

• **Mobmulti:** attiva il modo multicolore per uno sprite e fissa i tre colori comuni a tutti gli sprite multicolore. La sintassi è mobmulti sn, c1, c2, c3 dove sn è il codice dello sprite mentre i tre parametri successivi sono i codici dei tre colori.

• **Mobmove:** muove uno sprite tra due posizioni. È una istruzione piuttosto comoda anche se non potente come altre presenti in alcune espansioni che permettono di muovere gli sprite da interrupt. La sintassi è mobmove sn, x1, y1, x2, y2, v. I primi cinque parametri sono il codice dello sprite e le coordinate dei due punti fra cui questo deve essere mosso. L'ultimo parametro specifica la velocità di movimento: v=0 velocità massima e v=255 velocità minima.

• **Sprite:** permette di attivare/disattivare uno sprite. La sintassi è sprite sn, c dove sn è il codice dello sprite mentre c ha il seguente significato:

c=0: disattiva lo sprite
c=1: attiva lo sprite

Musica

• **Volume:** fissa il volume. La sintassi è volume n dove n può assumere un qualunque valore intero fra 0 e 15.

• **Envelope:** permette di settare i parametri attack, decay, sustain e release di una delle tre voci a disposizione. La sintassi è envelope v, a, d, s, r. Il primo parametro rappresenta la voce mentre i quattro parametri successivi sono i valori delle grandezze citate in precedenza. Il parametro v deve essere un intero fra zero e due mentre gli altri quattro parametri possono variare tutti fra 0 e 15.

• **Wave:** fissa la forma d'onda

per una voce. La sintassi è wave v, w. Il primo parametro è la voce mentre il secondo parametro ha il seguente significato:

w=16: onda triangolare
w=32: onda a dente di sega
w=64: onda quadra
w=128: rumore bianco

• **Play:** suona una nota da una voce. La sintassi di questa istruzione è duplice: play v, f, l oppure play v, "stringa-note", l. Nel primo caso il significato dei parametri è il seguente: v rappresenta la voce, f la frequenza della nota da emettere e l la lunghezza del tono. Nel secondo caso "stringa-nota" rappresenta la sequenza di note, da specificare nella notazione indicata sulla guida dell'utente, da emettere.

Aiuto alla programmazione

• **Renumber:** si tratta dell'arcinota e utilissima istruzione per renumerare le linee di un programma Basic. La sua sintassi è renumber s, in dove s rappresenta il numero da assegnare alla prima linea di programma e in l'incremento fra le linee.

• **Delete:** un'altra istruzione comodissima. Permette infatti di cancellare una o più linee (consecutive) di programma. La sintassi è delete i, f dove i e f sono rispettivamente la prima e l'ultima linea del blocco da cancellare.

• **Auto:** attiva la numerazione automatica delle linee. La sintassi è auto n dove n rappresenta l'incremento fra i numeri di linee da generare. Per disattivare la numerazione automatica dovete usare l'istruzione auto 0.

• **Find:** forse è l'istruzione più utile di questo gruppo. Permette infatti di cercare una stringa qualsiasi all'interno del programma Basic in memoria.

La sintassi è find stringa. Tenete presente che la stringa da cercare va specificata esattamente così come appare nel listato e che fra l'inizio della stringa e la d di find non ci devono essere spazi (verrebbero considerati come parte della stringa). Per esempio, se volete trovare le stringhe sys 49152 e "pippo" dovete usare le seguenti istruzioni:

```
findsys 49152
find"pippo"
```

• **Merge:** carica un programma e lo fonde con quello eventualmente in memoria. La sintassi è identica a quella dell'istruzione Load del Basic standard. Tenete presente che il primo numero di linea del programma da caricare deve essere maggiore dell'ultimo numero di linea del programma in memoria (usate Renumber per preparare il programma alla fusione).

• **Save:** è l'estensione dell'istruzione Save del Basic standard. Quindi oltre al formato usuale, Save potrà essere usata anche in questo modo: save "nomefile", ndv, is, init, fine. Il primo parametro è il nome del file. I due parametri successivi sono i soliti numeri di device e l'indirizzo secondario (da porre sempre uguale a zero per salvare un programma Basic). Gli ultimi due parametri permettono di specificare il blocco di memoria da salvare; rappresentano infatti il primo e l'ultimo indirizzo del segmento da salvare.

• **@:** permette di inviare un comando a una periferica. La sintassi è @ ndv, stringa-comando. Il primo parametro rappresenta il numero della periferica mentre il secondo il comando. Per esempio con @ 8, "$" potete visualizzare la directory del disco senza cancellare il programma in memoria (il list dei programmi può essere bloccato premendo il tasto Shift. Fate anche attenzione al modo in cui viene presentato il contenuto del dischetto).

• **Print@:** permette di visualizzare una stringa in una particolare posizione. La sintassi è print@ l, c, "stringa". I primi due parametri rappresentano la linea e la colonna da cui iniziare a visualizzare la stringa mentre l'ultimo parametro è la stringa da visualizzare. I parametri l e c devono essere compresi rispettivamente nei range 0-24 e 0-39.

• **Restore:** permette di modificare a piacimento il puntatore alla linea data corrente. La sintassi è restore ln dove ln è la linea cui deve fare riferimento il puntatore alla linea data.

• **Doke:** si tratta di un'istruzione molto comoda anche se raramente presente nelle espansioni. Consente infatti di effettuare un doppio poke. La sintassi è doke loc, val e funziona in questo modo: pone il byte basso di val nella locazione loc e il byte alto di val nella locazione loc+1.

• **Allclose:** chiude tutti i canali aperti con open. Questa istruzione non necessita di alcun parametro.

• **Inkey:** ecco un'altra istruzione potente e utilissima. Inkey praticamente è la versione potenziata dell'istruzione input del Basic standard perché permette di realizzare facilmente routine di input controllato. La sintassi dell'istruzione è inkey l, c, maxc, a$, r$(, fl). I primi due parametri rappresentano la linea e la colonna in cui porre il cursore e quindi devono essere compresi rispettivamente nei range 0-24 e 0-39. Maxc indica il numero massimo di caratteri accettabili e deve essere compreso fra 0 e 39-s (l'input deve quindi stare su una sola linea). Il parametro a$ è una stringa che contiene tutti e solo i caratteri da accettare in input. Il parametro r$ è la variabile stringa in cui verrà memorizzato il risultato dell'input. L'ultimo parametro, opzionale, permette di specificare il tasto che fa terminare l'input (normalmente tale tasto è il Return) e deve essere una variabile numerica. Ecco i valori che può assumere:

fl=-1: il tasto Crsr up pone fine all'input
fl= 0: il tasto Return pone fine all'input
l= 1: il tasto Crsr down pone fine all'input

Durante l'input potete usare i tasti di movimento del cursore, Delete e Insert.

• **Hardcopy:** stampa il contenuto del video in bassa risoluzione o il contenuto della pagina grafica. La sintassi è hardcopy n dove n può assumere i seguenti valori:

n=0: stampa il contenuto del video in bassa risoluzione
n=1: stampa il contenuto della pagina grafica

• **Case error goto:** permette di saltare a una determinata linea di programma se l'interprete rileva la presenza di un errore durante l'esecuzione del programma.
La sintassi è identica a quella dell'istruzione Goto del Basic standard.
Tenete presente che questa istruzione funziona solo se precedentemente è stata eseguita l'istruzione no error.
Il trap degli errori viene automaticamente disabilitato quando si torna in modo diretto.

• **Resume:** permette di specificare cosa deve fare l'interprete dopo aver rilevato un errore.
La sintassi è duplice: resume oppure resume next. Nel primo caso l'interprete esegue nuovamente l'istruzione che ha causato l'errore mentre nel secondo caso esegue l'istruzione immediatamente successiva. I due programmini successivi dovrebbero chiarire il funzionamento di questa istruzione:

```
10 no error: case error goto 100
20 b=0: b=10/b: print"b="b
30 end
100 b=1:resume
```

```
10 no error: case error goto 100
20 b=0: b=10/b: print"b="b
30 end
100 b=1: resume next
```

Il primo programma visualizza:

b=10 mentre il secondo b=1.

• **Usr:** è una istruzione abbastanza insolita perché permette di fare molte cose diverse fra loro. La sintassi è usr (n) dove n può assumere i seguenti valori:

- *n=0:* Usr fornisce il codice dell'errore incontrato dall'interprete. La **tavola 2** (elenco dei codici degli errori rilevabili dall'interprete Basic) contiene la lista completa dei codici associati ai possibili errori.
- *n compreso fra 8 e 20:* Usr legge il codice d'errore del drive.
- *n maggiore di 20:* Usr funziona come un doppio peek.

Vediamo qualche esempio d'uso di questa singolare istruzione:

```
a=usr(0): print a
```

questa istruzione visualizza il codice dell'errore incontrato dall'interprete e lo memorizza nella variabile a.

```
er=usr(8): if er=0 then print"Il drive è ok"
```

questa istruzione legge il codice d'errore del drive e visualizza il drive: è tutto ok se tale codice è uguale a 0.

```
ba=usr(43): print"basic start: "ba
```

in questo caso Usr compie la seguente operazione:

ba=peek(43)+peek(50)*256 e poi visualizza il contenuto di ba.

Daniele Maggio

Giocare da immortali

I tips di questo mese sono tanti e preziosi, come sempre. Accanto a due utility veramente originali ci sono, udite udite, ben tre routine per regalarvi vite e risorse illimitate in altrettanti videogame di successo

Le due utility, preziosissime, permettono di aumentare incredibilmente le prestazioni del drive e di rendere facilissima la fusione di due o più programmi Basic.

Turbo Load

Chi usa spesso il Commodore 64 avrà sicuramente sviluppato una certa insofferenza per la lentezza del drive 1541. Questo utilissimo e insostituibile dispositivo pur essendo nettamente più veloce del registratore non offre prestazioni entusiasmanti. Certo, le cose sono completamente diverse se potete disporre di una cartuccia acceleratrice o se, addirittura, avete lo Speed Dos oppure il Turbo Dos. In questo caso le prestazioni del 1541 diventano di tutto rispetto. Se però non avete questi strumenti potrete finalmente migliorare la situazione utilizzando Turbo Load. Questa utility è un po' lunga, per la verità, ma garantisce risultati incredibili: è in grado di quintuplicare la velocità di caricamento dei file programma! Per utilizzare la routine caricatela con LOAD"TURBO LOAD" e date il Run. Installata la routine, provate pure a caricare, con un normale Load, un programma lungo e... tenetevi ben saldi sulla sedia: il vostro drive farà veramente scintille. Per usare Turbo Load non è quindi necessario utilizzare istruzioni particolari ma basta usare Load nel modo consueto. Tenete presente che i programmi molto lunghi possono cancellare Turbo Load. In questo caso, ovviamente, sarete costretti a riattivare l'utility.

Fast merge

Programmando in Basic si ha la necessità di realizzare un gran numero di subroutine che poi devono essere integrate nel programma principale. Questa tecnica, necessaria per affrontare i problemi più difficili con sistematicità, comporta però un grosso problema alla fine: l'integrazione delle routine stesse.

Listato 1.

Nebulus

```
1 rem ------
2 rem -          -
3 rem - nebulus -
4 rem -          -
5 rem ------
6 :
10 forx=2816 to 2852
20 read b:pokex,b:c=c+b:next
30 if c<>4208 then print"<CLEAR>data error":end
50 print:print"unlimited lives y/n"
60 get t$:if t$<>"y" and t$<>"n" then 60
70 if t$="y" then poke 2841,8
80 print:print"infinite time y/n"
90 get w$:if w$<>"y" and w$<>"n" then 90
100 if w$="y" then poke 2846,0
110 print:print"insert tape please"
120 sys 2816
130 :
140 :
150 rem  ml data
160 :
170 :
180 data 32,44,247,32,108,245,169,76
190 data 141,209,3,169,24,141,210,3
200 data 169,11,141,211,3,76,176,2
210 data 169,82,141,212,128,169,1,141
220 data 137,182,76,0,128
```

Listato 2.

Out Run

```
1 rem ———
2 rem -         -
3 rem - out run -
4 rem -         -
5 rem ———
6 :
20 sys 65371:printchr$(5)"insert out run"
30 fora=53216 to 53255:read b:poke a,b:next
40 input"return to load":load
50 sys 53216
60 :
70 :
80 rem ml data
90 :
100 :
110 data 32,44,247,32,108,245,169,203
120 data 141,208,8,169,247,141,209,8,76
130 data 16,8,169,0,141,200,4,169,208
140 data 141,201,4,76,0,4,169,173,141
150 data 140,135,76,157,148
```

Lavorando con il Basic standard bisogna letteralmente ricorrere a virtuosismi per raccogliere le routine perché non sono assolutamente disponibili strumenti per automatizzare l'operazione o quantomeno per agevolarla. L'unico modo per aggirare l'ostacolo consiste nell'usare la solita espansione che mette a disposizione la comodissima istruzione merge per fondere tra loro più programmi Basic. Ma a questo punto c'è un altro inconveniente, forse maggiore di quello originario, dovuto al fatto che spesso l'espansione non può stare in memoria perché è incompatibile con alcune routine in Linguaggio macchina usate nel programma. Fast merge è la soluzione ideale a questo tipo di problema che affligge la stragrande maggioranza dei programmatori Basic. Fast merge è una routine completamente in Linguaggio macchina e completamente rilocabile che vi permetterà di fondere in memoria più programmi Basic. Per utilizzare la routine dovete copiare il listato corrispondente e dare il consueto Run. Per default la routine viene allocata a partire dalla locazione 49152, ma se lo desiderate potete modificare questo indirizzo solo cambiando il valore della variabile sa alla linea 30. Appena lanciata, Fast merge provvede a visualizzare il messaggio di installazione che notifica l'indirizzo di allocazione della routine in Linguaggio macchina vera e propria. La routine unisce il programma in memoria con uno presente su disco e va utilizzata nel modo seguente:

SYS 49152,"nomefile",8

Nomefile rappresenta il nome del programma che deve essere unito a quello in memoria. Naturalmente se avete rilocato la routine dopo sys dovrete mettere il nuovo indirizzo di allocazione. Tenete presente che i numeri di linea dei due programmi devono essere compatibili, cioè dovete fare in modo che l'ultimo numero di linea del programma in memoria sia minore del primo numero di linea del programma su disco. In caso contrario potrebbero verificarsi spiacevoli inconvenienti.

Listato 3.

Solomon's key

```
1 rem ————
2 rem -               -
3 rem - solomon's key -
4 rem -               -
5 rem ————
6 :
10 fora=544 to 593
20 read b:pokea,b:c=c+b:next
30 if c<> 4899 then print"<CLEAR>data error":end
50 print"unlimited lives y/n"
60 get t$:if t$<>"y" and t$<>"n" then 60
70 if t$="y" then poke 577,169
80 print:print"infinite time y/n"
90 get w$:if w$<>"y" and w$<>"n" then 90
100 if w$="y" then poke 582,96
110 print:print"unlimited fireballs y/n"
120 get f$:if f$<>"y" and f$<>"n" then 120
130 if f$="y" then poke 587,0
230 print:print"insert tape"
240 sys 544
250 :
260 :
270 rem ml data
280 :
290 :
300 data 32,44,247,32,108,245,169,51
310 data 141,142,8,169,2,141,143,8
320 data 76 ,16,8,169,64,141,91,1
330 data 169,2,141,92,1,76,234,3
340 data 169,198,141,165,8,169,248,141
350 data 30,19,169,15,141,165,56,76
360 data 15,8
32076 ,16,8,169,64,141,91,
```

Listato 4.

Merge

```
1 rem ——
3 rem - merge -
5 rem ——
6 :
10 rem
12 rem
20 poke53280,0:poke53281,0:print"<CLEAR><LT GREY><CTRL-N><CTRL-H>Loding data..."
30 sa=49152
40 v1=sa+96:h1=int(v1/256):l1=v1-h1*256:v2=sa+197:h2=int(v2/256):l2=v2-h2*256
50 fori=satosa+234:readd:pokei,d:next
70 pokesa+28,l1:pokesa+33,h1:pokesa+38,l2:pokesa+43,h2
80 print"<CLEAR>Merge installato da"sa:end
90 rem
91 rem — linguaggio macchina —
92 rem
100 data032,231,255,173,002,003,141,235
110 data192,173,003,003,141,236,192,173
120 data044,003,141,237,192,173,045,003
130 data141,238,192,169,096,141,002,003
140 data169,192,141,003,003,169,197,141
150 data044,003,169,192,141,045,003,032
160 data212,225,165,186,201,008,144,091
170 data169,001,166,186,160,003,032,186
180 data255,032,192,255,032,175,245,162
190 data001,032,198,255,032,228,255,032
200 data228,255,032,183,255,201,000,208
210 data058,169,202,160,192,032,030,171
220 data162,001,032,198,255,032,228,255
230 data032,228,255,032,183,255,201,000
240 data208,040,032,228,255,133,020,032
250 data228,255,133,021,169,005,141,239
260 data192,032,228,255,172,239,192,153
270 data251,001,201,000,240,047,238,239
280 data192,208,238,169,211,160,192,032
290 data030,171,173,235,192,141,002,003
300 data173,236,192,141,003,003,173,237
310 data192,141,044,003,173,238,192,141
320 data045,003,169,001,032,195,255,032
330 data204,255,076,116,164,132,011,032
340 data019,166,076,167,164,169,001,076
350 data049,243,013,077,069,082,071,073
360 data078,071,000,013,063,070,073,076
370 data069,032,078,079,084,032,070,079
380 data085,078,068,032,032,069,082,082
390 data079,082,000
```

Nebulus, Out Run, Solomon's key

Nella maggior parte dei giochi ci sono livelli veramente impossibili, anche per chi passa molte ore con il joystick in mano.

In questi casi riuscire ad avere la meglio sui nemici elettronici può diventare anche una questione di vita o di morte.

Sono pochissime infatti le persone che si arrendono davanti a un videogame quando il grado di difficoltà del gioco raggiunge i livelli più alti. I Super trainers sono tre routine che faranno la felicità dei giocatori più accaniti perché garantiscono vite e risorse illimitate in tre famosi videogame. I videogame in questione sono: Out run, Solomon's key e Nebulus.

Per utilizzare una delle tre routine dovete copiare il listato corrispondente e dare il Run. Tutte le routine provvederanno automaticamente a indicarvi le operazioni che dovete eseguire. Vediamo un po' più da vicino cosa fanno le routine in questione.

Il trainer di Out run vi permetterà di avere un tempo illimitato per terminare la gara, questo significa che finalmente potrete andare sempre a tutto gas senza preoccuparvi troppo di non fare incidenti. Oppure potrete andare a passo di lumaca e ammirare per quanto tempo volete il paesaggio circostante. Il trainer di Solomon's key vi fornirà invece tempo infinito, vite infinite e anche un numero illimitato di fireball. Questo significa, è facile da immaginare, che finalmente potrete completare ogni livello senza faticare assolutamente e magari divertendovi a eliminare i mostriciattoli più antipatici (c'è solo l'imbarazzo della scelta). Anche il trainer di Nebulus vi consentirà di completare ogni livello senza più faticare perché vi mette a disposizione infinite vite e tempo infinito.

Daniele Maggio

Continua il seguitissimo corso di Linguaggio macchina. Questa volta puntiamo la nostra attenzione sull'interrupt

Dritto al cuore del sistema

DRIVE O TAPE

Lo scopo di questa serie di articoli è di svelare tutto ciò che è legato ad applicazioni grafiche particolari che rendono unico il C64. Questi argomenti sono di estremo interesse per gli amatori del Linguaggio macchina e utilissimi per chi vuole imparare a programmare seriamente.

Parliamo di interrupt

Quando accendiamo il C64 la nostra attenzione è subito attirata da un quadratino lampeggiante, meglio conosciuto come cursore.

Forse non vi siete mai chiesti cosa stia facendo il computer e perché riesca a far lampeggiare il cursore ininterrottamente anche se proviamo a scrivere qualcosa oppure a muoverci sullo schermo. In pratica è come se avesse due programmi in esecuzione contemporanea, uno per ricevere informazioni dalla tastiera e uno per far lampeggiare il cursore. In realtà la faccenda è un po' diversa. Esistono effettivamente due programmi, ma è lo stesso microprocessore che li esegue alternativamente a una velocità tale che l'osservatore umano ha l'impressione di assistere all'esecuzione continua e contemporanea di due programmi e potrebbe immaginare debbano esserci due processori distinti. Quando il microprocessore arresta l'esecuzione di uno dei due programmi (programma principale) per eseguire l'altro, si dice che avviene un'interruzione, o interrupt. Questi interrupt possono essere di vario tipo: hardware oppure software. Per i più esperti possiamo aggiungere che il microprocessore reagisce in un modo tutto suo a una richiesta di interruzione. Prima di tutto il 6510 completa l'istruzione che stava eseguendo, poi controlla se il flag I del registro di stato è settato, cioè contenente uno. In caso affermativo riprende il suo compito ignorando la richiesta di interrupt altrimenti salva nello stack il byte alto e basso del program counter e il registro di stato. Infine il program counter viene caricato con i valori contenuti in $FFFE e $FFFF, che costituiscono il vettore di interrupt.Ciò che abbiamo descritto è un interrupt chiamato Irq, che è controllabile tramite un opportuno flag del registro di stato. (l'istruzione SEI di codice $78 pone a uno il flag I, mentre la CLI di codice $58 setta lo stesso flag a zero).

Esiste anche un altro tipo di interrupt non mascherabile a priorità più alta chiamato Nmi. Quando si verifica questo tipo di interrupt non è possibile farlo ignorare alla Cpu. Il tasto Restore, per esempio, è collegato direttamente alla linea di Nmi, quindi a ogni sua pressione viene generato un Mni. Per l'Nmi il vettore è allocato a $FFFA $FFFB.

Tralasciamo l'Nmi e interessiamoci dell'Irq.

Se andiamo a leggere le locazioni del vettore con PRINT PEEK (65534) e PRINT PEEK (65535) avremo l'indirizzo della routine di gestione degli interrupt. Questa routine è allocata nella Rom kernal a $FF48 e consente all'hacker, termine che spetta di diritto a ogni utente del C64 padrone di queste tecniche, di inserire una propria routine di interrupt al posto di quella di sistema.

Il **listato 1** riporta la routine di sistema. In generale basta sapere che questa routine salva nello stack i tre registri (memorie personali del 6510), verifica la condizione del flag B del registro di stato per sapere se è un interrupt software o no.

Se l'interrupt è hardware salta indirettamente alla locazione puntata dal vettore allocato in $0314 e $0315. Si osservi che la locazione $0314 (788) contiene il byte basso della routine di interrupt, mentre $0315 (789) il byte alto, ma soprattutto che queste due locazioni si trovano in Ram e quindi modificabili a piacere.

Ora siamo in grado di capire come fa il computer a far lampeggiare il cursore. A intervalli regolari, scanditi dal timer A del Cia 6526 che è uno dei dispositivi di input/output, viene provocato un interrupt.

La Cpu blocca ciò che stava facendo, salta a una particolare locazione puntata da $0314 $0315, esegue un programma che fa lampeggiare il cursore e alla fine riprende il suo vecchio compito.

L'intervallo di tempo a cui avvengono queste chiamate è normalmente di un 60esimo di secondo (1/60 di secondo viene chiamato Jiffy), ma può essere variato. Si provino le seguenti istruzioni:

```
POKE 56325,X  con 0<X<256.
```

Valori molto piccoli fanno letteralmente volare il cursore.

Il valore standard è 66, comunque a ogni pressione di Run/Stop e Restore il valore standard viene ripristinato.

La routine che fa lampeggiare il cursore, che aggiorna l'orologio interno e fa altre cosucce, si trova a $EA31. Resta ora da chiarire come far riprendere alla Cpu i suoi vecchi compiti, cioè far terminare l'interrupt. Per far ciò esiste un'istruzione chiamata Rti dall'inglese Return From Interrupt (codice $40). Questa istruzione riprende dallo stack il valore del registro di stato e ristabilisce il vecchio indirizzo del program counter.

È inoltre necessario recuperare anche il valore dei registri della Cpu. Parte della routine che comincia ad $EA31 ha anche questo compito.

```
$EA81 68 PLA
$EA82 A8 TAY
$EA83 68 PLA
$EA84 AA TAX
$EA85 68 PLA
$EA86 40 RTI
```

Si osservi il passaggio della cima dello stack all'accumulatore (istruzione Pla).

Curiosità

Quando si da il Run a un programma in Basic il cursore sparisce.

Listato 1.
Routine di sistema per la gestione dell'interrupt.

```
$FF48    PHA
$FF49    TXA
$FF4A    PHA
$FF4B    TYA
$FF4C    PHA
$FF4D    TSX
$FF4E    LDA $0100+4,X
$FF51    AND #$10
$FF53    BEQ $FF58
$FF55    JMP ($0316)   ;vettore in Ram per Brk
$FF59    JMP ($0314)   ;vettore per Irq standard
```

Significa che non avvengono più interrupt? Ovviamente la risposta è no! Provate il seguente programma:

```
10 POKE 204,0
20 GET A$:IF A$="" THEN 20
30 PRINT A$;
40 GOTO 20
```

Avete visto che il cursore lampeggia ancora?

Ricapitolando

Proviamo a fare una nostra routine di interrupt (**listato 2**).

Date Run e poi Sys 49152. Vediamo di capirne il funzionamento.

La prima istruzione (SEI) blocca eventuali richieste di interrupt, viene poi modificato il vettore di interrupt affinché punti a $C00D e alla fine gli interrupt sono riattivati (Cli). È bene notare che durante la modifica del vettore di interrupt è indispensabile disattivare ogni possibile Irq. La nuova routine di interrupt stampa sullo schermo una serie di caratteri, alternando la loro immagine in reverse a quella normale, provocando così il fenomeno del lampeggiamento.

Il programma è allora composto da due parti:

1) si occupa di settare il nuovo vettore di interrupt;

2) routine che gestisce il nuovo interrupt, provvedendo a richiamare anche la vecchia routine di sistema ($EA31).

Ancora qualche precisazione: i vari dispositivi di I/O sono collegati tramite una linea hardware al piedino di Irq del 6510. Questi dispositivi li possiamo dividere in due categorie: il Cia e il Vic II. I chip Cia (Complex Interface Adapter) sono due: uno di essi (Cia numero 1, i cui registri si trovano da $DC00 a $DC0F), collegato al pin di Irq della Cpu, è usato per le funzioni di sistema come scansione della tastiera, aggiornamento dell'orologio interno, lettura dei joystick, delle paddle e della penna ottica. L'altro, il Cia numero 2, collegato al pin di Nmi della Cpu, si occupa delle funzioni dell'utente, come controllo della porta Rs232 e della porta seriale (Cia numero 2 registri da $DD00 a $DD0F). Il Cia numero1 è responsabile degli interrupt che si verificano ogni 60esimo di secondo, ma è facile bloccarne la sorgente di interrupt mantenendo contemporaneamente possibili richieste da altri dispositivi. Questo si ottiene con:

```
LDA #$7F
STA $DC0D
```

non fatelo da Basic altrimenti si blocca tutto.

La procedura appena illustrata è molto importante perché permette di evitare chiamate che sottraggono tempo prezioso alla Cpu.

Infatti uno degli svantaggi derivanti da un uso intensivo degli interrupt è che si verificano delle perdite di tempo ogni volta che avvengono, tempo che è sfuttato

Listato 2.
Interrupt personalizzato (con rispettiva versione Basic).

```
$C000    SEI
$C001    LDA #$0D
$C003    STA $0314
$C006    LDA #$C0
$C008    STA $0315
$C00B    CLI
$C00C    RTS
$C00D    LDX #$05
$C00F    LDA $C025,x
$C012    EOR #$80
$C014    STA $C025,x
$C017    STA $0400,x
$C01A    LDA #$07
$C01C    STA $D800,x
$C01F    DEX
$C020    BNE $C010
$C022    JMP $EA31
$C025 00 08 05 0C 0C 0F

In Basic si ha (Prog 1 sulla cassetta):

 5 poke 56325,20
10 for t=49152 to 49152+42
20 read a:poke t,a
30 next t:end
40 data 120,169,13,141,20,3,169,192,141,21,3,88
50 data 96,162,5,189,37,192,73,128,157,37,192,157
60 data 0,4,169,7,157,0,216,202,208,237,76,49,234
70 data 0,8,5,12,12,15
```

per salvare i registri nello stack.

Oltre ai Cia, anche il Vic II (siglato 6567) è in grado di generare degli interrupt al verificarsi di particolari eventi. Ci si può trovare in una situazione nella quale gli interrupt del Vic II devono essere gestiti senza alcun ritardo. In questo caso, affinché non si verifichino attese, oppure il Cia numero1 non richieda l'attenzione proprio quando ne ha bisogno il Vic II, è indispensabile bloccare il Cia numero 1 come visto sopra. Ci sarebbero tante altre cose da dire a proposito dei dispositivi che generano gli interrupt, ma noi ci dedicheremo solo al Vic II.

Interrupt e collisioni

Il chip Vic II (Video Interface Chip), è un vero e proprio gioiello nel gestire la grafica. Le sue doti maggiori sono la possibilità di gestire fino a otto sprite hardware. Se avete provato il programma Finestra Hardware (comparso sul numero di ottobre 1989 e ripetuto sui due numeri seguenti) ne avrete contati anche 66. Questa volta ci occuperemo proprio degli sprite, ma per ottenerne un numero così elevato dovrete pazientare ancora un pochino.

In molti giochi, che anche i più esperti programmatori guardano sempre con curiosità, siamo al comando di una navicella simulata (sprite), ma succede che l'alieno di turno ci colpisca con i suoi proiettili mortali (sprite oppure caratteri), provocando appunto la nostra ingloriosa fine. Il programma è allora in grado di stabilire quando due sprite, oppure uno sprite e un carattere si scontrano.

Questo è reso possibile dalla manipolazione di alcuni registri del Vic II (locazioni di memoria che si trovano a partire da $D000 fino a $D02E). Dando per scontato che tutti sappiamo a grandi linee cosa sono sprite e caratteri, vale la pena ricordare che gli sprite sono indipendenti da tutto ciò che compare sullo schermo, cioè su di essi non possiamo agire con i tasti cursore come siamo soliti fare con gli altri oggetti (caratteri) che si trovano sullo schermo.

Lo sprite è una matrice di pixel che si sovrappone allo schermo, ma è come se non ne facesse parte. Quando due sprite si toccano, cioè quando le figure rappresentate dagli sprite vengono a contatto, il Vic II è in grado di richiedere l'attenzione alla occupatissima Cpu, provocando un segnale di interrupt. Perché ciò avvenga è necessario eseguire una procedura di inizializzazione del Vic II. I registri che ci interessano sono due: locazione $D019 (53273) e $D01A (53274), descritte nella **tavola 1**.

Queste due locazioni hanno funzioni diverse a seconda che vi si scriva qualcosa (Poke) oppure che vi si legga (Peek).

Scrivendo nella locazione $D01A si attivano le possibili sorgenti di interrupt che il Vic II

Listato 3.
Routine di inizializzazione per la gestione degli interrupt.

```
$C000 SEI
$C001 LDA #$04
$C003 STA $D01A                 ;attiva interrupt Vic II
$C006 LDA #$12                  ;modifica il vettore di interrupt
$C008 STA $0314                 ;in modo da farlo puntare alla
$C00B LDA #$C0                  ;routine di gestione
$C00D STA $0315
$C010 CLI                       ;ripristina ogni tipo di Irq
$C011 RTS

Routine di gestione e riconoscimento Irq

$C012 LDA $D01E                 ;resetta collision
$C015 LDA $D019                 ;legge tipo
$C018 STA $D019                 ;resetta
$C01A AND #$04
$C01C BNE $C022                 ;salta se gli sprite sono in contatto
$C01F JMP $EA31                 ;torna a routine di sistema e chiude
$C022 LDX #$FF                  ;segue lampeggio schermo
$C024 LDY #$FF
$C026 INC $D020
$C029 DEX
$C02A BNE $C026
$C02C DEY
$C02D BNE $C026
$C02F JMP $EA31
```

possiede.

Per attivare gli interrupt desiderati è necessario porre a uno il bit interessato. In lettura ci dice quali sono gli interrupt attivati. Facciamo un esempio: se si vogliono provocare degli interrupt ogni volta che uno sprite si scontra con un carattere è necessario attivare il bit 1.

In Basic si procede così:

```
POKE 53274,PEEK(53274) or (2^1)
```

In Assembler invece:

```
LDA $D01A
ORA %00000010  ;numero binario pari a #$02
STA $D01A
```

Se invece si vuole disattivare lo stesso tipo di interrupt bisogna porre a zero il bit desiderato.

In Basic:

```
POKE 53274,PEEK (53274) and (255-2^1)
```

In Assembler:

```
LDA $D01A
AND %11111101  ; in esadecimale #$FD
STA $D01A
```

La locazione $D019, che ha i bit con lo stesso significato visto per la locazione $D01A, ha invece delle funzioni diverse in lettura e in scrittura.

In lettura lo stato alto di un bit implica che è avvenuto l'interrupt a cui quel bit si riferisce. Il bit sette di questa locazione viene sempre impostato a uno indipendentemente dal tipo di interrupt verificatosi. Esempio: due sprite si sono toccati, la locazione $D019 conterrà il valore $84 (132), cioè %10000100.

La coppia Cpu e Vic II gestisce gli interrupt in un modo un po' particolare. Infatti tutte le volte che si verificano degli interrupt del Vic II, il chip grafico aspetta conferma da parte della Cpu dell'avvenuto riconoscimento dell'Irq. Se la Cpu non manda questo segnale di ritorno il Vic II non può più generare interrupt.

La Cpu manda conferma che l'Irq è stato assolto scrivendo uno nei bit della locazione $D019 corrispondenti alla causa di interrupt. Ricapitolando: scrivendo nella locazione $D019 si resettano le sorgenti di interrupt, permettendo che ne possano avvenire degli altri. Per resettare la sorgente bisogna porre a uno il bit a cui si riferisce l'interrupt.

Questo fatto è molto importante, perché se non avviene non si possono più verificare interrupt di quel tipo. È compito del programmatore riconoscere il tipo di interrupt che si è verificato e provvedere a resettare la sorgente. Ogni routine di gestione di interrupt del Vic II deve cominciare proprio nel seguente modo:

```
LDA $D019         ;legge il tipo di interrupt
STA $D019         ;resetta tutte le sorgenti
                  interessate
AND %00000100     ;And con $04: controlla il tipo di
                  interrupt
BNE go on         ;se è l'interrupt voluto esegue la
                  routine di gestione
JMP $EA31         ;altrimenti esegue la routine di
                  sistema
```

Questa serie di istruzioni si possono considerare come lo standard con cui iniziano le routine di gestione degli interrupt.

È bene osservare che gli interrupt del Vic II si comportano come dei normali Irq, dunque la routine che li vuole gestire deve modificare il vettore $0314 e

Listato 4.
Esempio di rilevamento collisioni sprite sprite senza interrupt.

```
1000 A=peek (53278)
1010 for T=0 to 7
1020 B=A and (2^T)
1030 if B>0 then print "sprite ";T;"è in collisione"
1040 next T
```

$0315.

Il **listato 3** riporta le routine di inizializzazione per gestire gli interrupt del Vic II. Si osservi il ruolo della locazione $D01A e si provi il programma in Basic (LOAD"PROG 2",8) che dopo aver posto in 49152 la precedente routine in Assembler permette di far muovere uno sprite (giallo) con i tasti cursore. Facendo scontrare lo sprite giallo con quello nero si provoca un interrupt, evidenziato dal lampeggiare dello schermo.

Vi sono altre due locazioni da ricordare perché permettono di sapere quali sprite sono interessati alla collisione. Ogni volta che due sprite collidono tra loro il Vic II setta a uno alcuni bit della locazione $D01E (53278); questi bit corrispondono ai vari sprite in collisione (bit zero sprite zero e così via). Il valore di questa locazione non muta fino a che non avviene una lettura tramite una Peek, dopodiché la locazione viene azzerata automaticamente, permettendo altri interrupt (è allora necessario preservarne il valore, perché una successiva lettura non riporterà lo stesso valore). Ciò significa che fino a quando non si legge la locazione $D01E non possono essere rilevate altre collisioni. La locazione $D01F (53279) si comporta allo stesso modo della $D01E, solo che viene impostata quando uno sprite urta un carattere.

Tavola 1.

Significato dei bit delle locazioni $D019 e $D01A.

bit 0:	*Irq Raster*
bit 1:	*Irq collisione sprite carattere*
bit 2:	*Irq collisione sprite sprite*
bit 3:	*Irq triggerato per penna ottica*
bit 4,5,6:	*non utilizzati*
bit 7:	*qualunque Irq del Vic II*

Il Vic II non fornisce alcuna informazione sul codice del carattere coinvolto.

Ovviamente le collisioni si possono gestire anche senza scomodare gli interrupt, basta leggere e quindi resettare di continuo i registri collisione $D01E e $D01F (**listato 4**).

Come sempre, però, se si vogliono i risultati migliori non si possono evitare gli interrupt e il Linguaggio macchina.

Il programma Prog 3 sulla cassetta è l'equivalente del programma 2, ma scritto interamente in Basic. La **tavola 2** riassume le operazioni da compiere per gestire le collisioni.

La prossima volta impareremo a gestire il bit zero della locazione $D01A, che viene chiamato raster.

Glossario

Microprocessore/Cpu/6510/6502 termini tra loro analoghi, usati per riferirsi al cervello del C64.

- **Program Counter/Pc**
  È un registro a 2 byte, a uso quasi esclusivo della Cpu, che contengono in ogni istante l'indirizzo di memoria della istruzione che la Cpu sta eseguendo. Sono l'analogo del numero di riga di un programma in Basic.

- **Stack**
  Area di memoria usata dalla Cpu per parcheggio di dati.

- **Registro di stato**
  È un registro che generalmente viene indicato con la lettera (P) maiuscola.
  Contiene le informazioni che riguardano lo stato del microprocessore.

- **Flag**
  Particolare bit di una locazione. A seconda del valore che essa assume la Cpu prende decisioni diverse.

- **Vettori**
  Sono due locazioni di memoria che forniscono un indirizzo a 16 bit.
  L'indirizzo si calcola nel modo seguente: contenuto loc. inferiore +(contenuto loc. maggiore)*256. Esempio: vettore 788 789. La locazione 788 contiene 15, mentre la 789 contiene 192. Il vettore punta a 15+192*256=49152+15.

Tavola 2.

Schema per gestire le collisioni.

1) Resettare registri collisione $D01E-$D01F.
Si ottiene con Peek (53278) e Peek (53279)
2) Scrivere routine di interrupt e attivare gli opportuni bit della maschera di interrupt ($D01A).
3) Dopo ogni collisione eliminare la causa della collisione e resettare nuovamente i registri collisione.

Nicola Chiminelli

(continua)

L'utility che pubblichiamo vanta alcune caratteristiche proprio innovative. Si tratta infatti di un programma per compattare file che, oltre a utilizzare un'algoritmo di compressione decisamente efficace, risulta comodissimo da usare grazie alla velocità di svolgimento del compito

Compatto è bello

Già da qualche anno, anche i piccoli computer come il C64 contano, fra la miriade di utility per essi create, molti programmi che permettono di comprimere file. Comprimere un file significa, in pratica, codificarlo secondo una procedura particolare, espressa da un algoritmo più o meno complesso, che riduce la quantità di memoria necessaria per contenere una catena di dati. Il risultato è un file più breve e maneggevole che contiene, naturalmente, la stessa quantità di informazioni. L'unico problema consiste nel restituire alla normalità la catena di dati, operazione necessaria per poter utilizzare le informazioni in essa contenute.

Il programma

L'utility che trovate sulla cassetta e che potete caricare con LOAD"FLASH PACK", è stata pensata per comprimere un file programma. L'uso richiede qualche informazione, ma è estremamente semplice. Vediamo quali sono le caratteristiche generali di Flash Pack: innanzitutto la velocità di compattamento. I migliori compattatori in circolazione impiegano qualche ora per effettuare compattamenti meno efficaci. E questo è appunto il secondo elemento fortemente a favore di Flash Pack: la capacità di ridurre notevolmente addirittura i file già compattati con altri packer meno efficienti.

Per gli esperti, Flash Pack offre la possibilità di comprimere in un unico file tutti i moduli di un programma. Infine, è in grado di compattare file lunghi ben 242 blocchi! Pensate che i migliori packer, con prestazioni analoghe a Flash Pack, impiegavano circa sei ore per comprimere file di poco più di 230 blocchi, che venivano ridotti a 180 circa, mentre la nostra utility comprime la bellezza di 242 blocchi in 314 secondi, riducendo il file a soli 160 blocchi!

I tempi di decompattamento, poi, sono eccezionali.

Per esempio, occorrono meno di tre secondi per decomprimere un file di 242 blocchi. Se non ci credete non dovete far altro che fare qualche prova seguendo le istruzioni dei prossimi paragrafi.

Figura 1. Il pannello di controllo del programma Flash Pack

Figura 2. Al termine del compattamento il programma informa sull'esito dell'operazione

Come funziona

Prima di lanciare Flash Pack, caricate il programma che volete comprimere. Questo è necessario per poter annotare l'informazione visualizzata dal comando List, che dovete digitare al termine del caricamento del programma da comprimere.

Il programma non deve essere in Basic, o meglio, deve essere tale per cui, se caricato ed esaminato con List, restituisca un messaggio del tipo:

10 SYS 2064

Prendete dunque nota del valore che segue la parola Sys. Il numero è in forma decimale, mentre il programma Flash Pack si aspetta che voi introduciate un numero esadecimale come indirizzo di partenza del programma. Digitate dunque il **listato 1**, salvatelo su disco, e lanciatelo. Digitate il valore decimale che avete annotato e vi verrà restituito il corrispondente valore esadecimale. Prendetene nota. Ora caricate e lanciate Flash Pack. Lanciato il programma compare la schermata riprodotta in **figura 1**. Il cursore lampeggia sulla destra della prima di quattro richieste di input. A questo punto dovete digitare il nome del programma da compattare. Alla seconda richiesta di input dovete digitare il nome che volete abbia il programma quando sarà compattato (infatti viene generato un nuovo file). Alla terza richiesta di input dovete inserire il valore esadecimale trovato nel modo illustrato sopra. La quarta richiesta di input attende l'inserimento di un altro valore esadecimale che rappresenta la condizione della memoria del C64 al momento del lancio del programma. Siccome qui ci occupiamo della compattazione di file programma dovete digitare il valore $37.

Fatto questo vi viene chiesto se volete disabilitare momentaneamente il video per incrementere la velocità di elaborazione del programma.

La richiesta successiva permette di scegliere fra due diversi modi di compattamento. La resa di ciascuno di essi dipende dalla struttura generale della catena di dati da comprimere. Fate qualche prova per rilevare eventuali differenze. Al termine dovete inserire nel drive il disco su cui si trova il file da leggere e comprimere (sorgente). Quando il compito del programma è terminato, esso vi invita a inserire nel drive il disco su cui scaricare il file compattato (destinazione).

Listato 1.

Programma per convertire numeri dal formato decimale a quello esadecimale.

```
5 print"<CLEAR>":dima$(16)
10 forx=0to15:reada$(x):next
100 input"numero";n
110 n4=n-int(n/16)*16:n=n-n4
120 n3=n-int(n/256)*256:n=n-n3
130 n2=n-int(n/4096)*4096:n=n-n2
140 n1=n-int(n/65536)*65536
150 print"$"a$(n1/4096);a$(n2/256);a$(n3/16);a$(n4)
200 goto100
500 data 0,1,2,3,4,5,6
505 data 7,8,9,a,b,c,d
510 data e,f
```

Programmi senza Sys

I programmi in Basic non possono essere lanciati mediante una Sys. A causa di questo, i programmi di questo tipo non possono essere compattati, a meno che non esista la possibilità di compilarli. Un file Basic compilato, infatti, restituisce un messaggio di Sys se caricato ed esaminato con List prima di lanciarlo.

Studio Bitplane

Simon says "Draw!"

A quanto pare la vecchia espansione Simon's Basic è ancora un punto di riferimento per quegli appassionati di Basic del C64 che vogliono sfruttare le capacità grafiche e sonore del proprio home. L'articolo che pubblichiamo illustra come utilizzare a fondo l'istruzione DRAW di questa espansione

Tra le innumerevoli possibilità e applicazioni offerte dal piccolo e forte Commodore 64, avrete sicuramente avuto modo di notare che le capacità grafiche sono veramente notevoli: esso possiede una pagina grafica formata da un rettangolo di 320 x 200 punti o pixel indirizzabili singolarmente: questo significa che ogni singolo punto può essere acceso o spento indipendentemente dallo stato degli altri.

Avrete sicuramente notato anche che la grafica non è tanto agevole da utilizzare, tanto che spesso si ricorre all'aiuto di pacchetti applicativi esterni per ovviare a questo problema.

Uno di quelli che ha avuto più fortuna è sicuramente il Simon's Basic, progettato per darvi la possibilità di accedere a tutta la potenza del vostro computer senza dover impazzire con metodologie complicatissime.

In questa sede però, non si vuole parlare del package applicativo nel suo complesso, quanto focalizzare l'attenzione su alcune istruzioni, spesso non adeguatamente trattate, che possono essere d'aiuto in svariate circostanze. Immaginate ora di dover creare un'immagine grafica molto complessa che non sia possibile realizzare utilizzando i simboli grafici di base della macchina.

Potreste muovervi in diverse direzioni: rimanendo nell'ambito del Simon's Basic (d'ora in avanti anche se non specificato esplicitamente si intenderanno sempre istruzioni e metodologie relative a questo pacchetto), se la vostra immagine ha dimensioni

Tavola 1.

Elenco dei valori nella stringa di DRAW e istruzioni relative.

0 ==> muovere un pixel a destra
1 ==> muovere un pixel in alto
2 ==> muovere un pixel in basso
3 ==> muovere un pixel a sinistra
5 ==> muovere un pixel a destra e tracciare un punto
6 ==> muovere un pixel in alto e tracciare un punto
7 ==> muovere un pixel in basso e tracciare un punto
8 ==> muovere un pixel a sinistra e tracciare un punto
9 ==> figura terminata

ridotte, è possibile creare un Mob (o sprite). Un Mob è un oggetto programmabile che può assumere una vasta gamma di forme.

Questo oggetto può muoversi su tutto lo schermo, dicendo semplicemente al computer dove volete che si trovi lo sprite in questione e con quale velocità deve spostarsi dalla posizione attuale.

Nel Basic standard la generazione e l'animazione degli sprite necessita di numerose istruzioni Poke, mentre con il Simon's Basic sono sufficienti semplici comandi: l'effetto è veramente notevole. Il problema però è che le massime dimensioni della figura devono essere di 24 pixel di larghezza e 21 di altezza, il che è però limitativo. Se il vostro disegno occupa dimensioni maggiori, come nella stragrande maggioranza dei casi, potreste ricorrere a uno stratagemma di questo tipo: associate una lettera o un numero a ogni tipo di spostamento e otterrete una sequenza che eseguita correttamente vi permetterà di realizzare il disegno con una buona precisione. Unico neo di questo procedimento, perfettamente attuabile anche con il Basic standard, è la lentezza del disegno, in quanto, prima di disegnare è necessario effettuare una sequenza di istruzioni che permetta di riconoscere il carattere da eseguire e per ciò dia indicazioni sulla direzione da prendere.

Un esempio è rappresentato dall'istruzione DRAW ed è proprio su questa istruzione che focalizzeremo la nostra attenzione.

L'istruzione DRAW

Mediante questa istruzione è possibile creare immagini grafiche di qualsiasi dimensione, con un'elevata velocità di esecuzione e soprattutto con una relativa facilità di spostamento su tutto lo schermo. Il comando DRAW vi permette di preparare una figura e quindi di visualizzarla sullo schermo. La figura viene composta come quando si disegna su di un foglio di carta, cioè senza staccare la matita dal foglio. Ma la notevole differenza sta proprio nel fatto che tramite quest'istruzione potrete anche spostare la matita senza disegnare e questo vi permette di realizzare qualsiasi cosa.

Il formato dell'istruzione è il seguente:

DRAW "nnnnnnnnn....9",x,y,plot type

dove x e y sono le coordinate del punto sullo schermo dove inizia la figura e plot type è un parametro che ha il seguente significato:

0 spegne un punto sullo schermo (Off)
1 accende un punto sullo schermo (On)
2 inverte un punto sullo schermo cioè lo accende se spento e lo spegne se acceso

La stringa invece è un insieme di numeri ognuno dei quali rappresenta un'istruzione per il Commodore 64 su come muovere la matita per disegnare la figura.

È evidente che al posto della stringa si può utilizzare il nome di una variabile a cui si è assegnato il valore mediante una linea di programma o una istruzione Data.

Su ogni linea di programma è possibile porre al massimo 74 istruzioni di questo tipo tra virgolette. È tuttavia possibile aggiungere delle stringhe di istruzioni fino a un massimo di 255.

Per continuare la figura oltre questo limite occorre definire una nuova origine dove è terminata la stringa precedente.

In **tavola 1** trovate l'elenco dei valori e l'istruzione associata.

Occorre ricordare di porre come ultimo carattere del disegno il 9 in modo da informare il C64 che il disegno è terminato. Da notare il fatto che facendo uso di queste poche istruzioni si possono ottenere tutte le figure grafiche possibili e immaginabili. La figura però non viene tracciata fino a quando i comandi DRAW e ROT non vengono eseguiti.

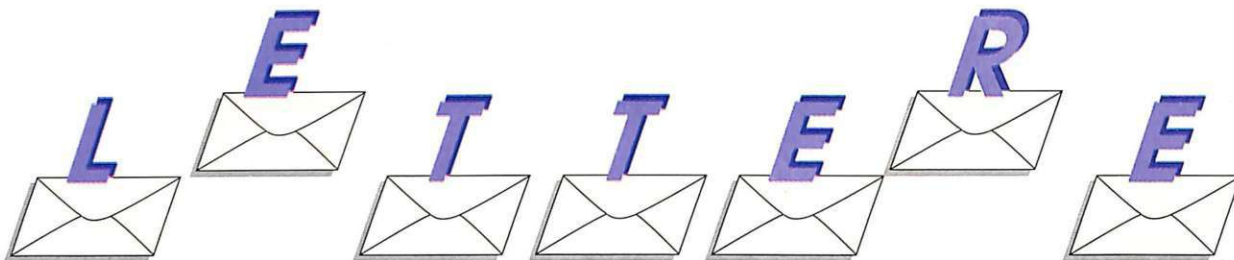
Il comando ROT vi permette di visualizzare una figura creata con DRAW specificando dimensioni e angolo di rotazione. Fate attenzione a non impostare dimensioni troppo grandi della figura perché se la stessa non ci sta nel video, non verrà rappresentata.

L'ultima cosa da dire riguarda la possibilità di spostare la figura sul video: occorrerà prima cancellare la figura esistente e ciò si ottiene ridisegnando con la stessa procedura ma con plot type uguale a zero.

Poi si variano le coordinate di inizio disegno e si disegna con plot type 1. La metodologia da seguire è la seguente: dovete eseguire il disegno su carta millimetrata o quadrettata e poi, immaginando che ogni quadretto sia un pixel, riempite tutti i quadratini in modo da avere un'idea di come verrà rappresentato per pter apporre le necessarie correzioni.

Poi, con santa pazienza, iniziate da un angolo e, utilizzando i codici sopra esposti, fornite al computer tutti i dati necessari per il disegno e create le stringhe necessarie. A questo punto non dovrete far altro che eseguire ROT e DRAW per ottenere il disegno.

Il corso di scacchi comparso sui numeri di *Radio Elettronica & Computer* nel 1988 utilizzava la tecnica illustrata: tutti i pezzi e la scacchiera erano ottenuti in questo modo e il risultato ripagava la fatica impiegata.

Alberto Palazzo

Effetti speciali

Siamo due ragazzi sedicenni e abbiamo comprato da poco tempo un Commodore 64. L'inverno è lungo e freddo e così passiamo parecchio tempo a giocare con questa bellissima macchina. Siccome desideriamo diventare degli esperti di programmazione vorremmo sottoporvi due domande che ci siamo posti proprio osservando i nostri videogame. Innanzitutto vorremmo sapere come fa il computer a gestire contemporaneamente tante cose come gli sprite dell'astronave in un gioco spaziale, lo sfondo, i proiettili, i nemici, il punteggio, i suoni e tutto il resto. Inoltre siamo estremamente curiosi di sapere come dobbiamo fare per scrivere un videogame.

Luca Passoni e **Sandro Bruni**
Milano

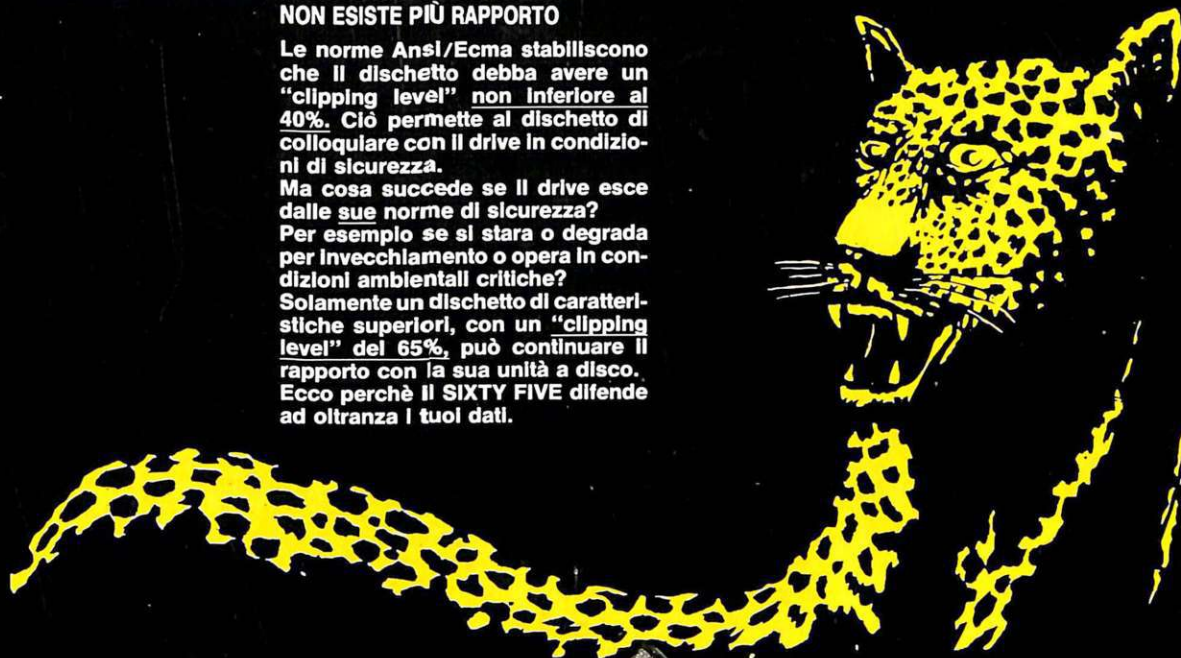
Diventare degli esperti di programmazione è un ottimo modo per riempire le ore delle grigie giornate invernali, sicuramente meglio che passare tutto il tempo davanti a un videogame o alla Tv. Naturalmente qualsiasi scienza (sviluppare programmi è una scienza) richiede studio, applicazione e metodo. A nessuno è dato di sapere senza avere imparato. La cosa migliore da fare per chi inizia a interessarsi ai computer è quella di avere un computer su cui sperimentare le nozioni acquisite.

Per rispondere alle domande da voi poste dobbiamo illustrare, prima di tutto, la struttura generale di un videogame. Un videogioco è un programma come qualsiasi altro, solo che fa un uso intensivo delle proprietà grafiche e sonore della macchina su cui gira. A parte qualche algoritmo che definisce la strategia di gioco, c'è sempre una serie di routine dedicate alla grafica e al suono. Un algoritmo è il concetto che determina la logica di funzionamento di un programma. Nel caso di un gioco spaziale, per esempio, l'algoritmo di strategia regola il numero di nemici da abbattere, la quantità di munizioni del mezzo spaziale, l'assegnamento dei punteggi, i livelli di gioco e così via. Ogni videogame è caratterizzato da una strategia e da uno scopo. Il programmatore, quindi, prima di cominciare a stendere il programma, deve definire lo schema generale del gioco. Per esempio deve immaginare che vi sia uno sprite che rappresenta l'astronave del giocatore, altri sprite che rappresentano i nemici e altri che definiscono i proiettili per abbattere i nemici. Lo sfondo deve muoversi orizzontalmente per dare l'impressione che siano l'astronave e i nemici a spostarsi. Tutti questi elementi devono essere coordinati da un algoritmo di strategia che rileva quando un proiettile ha colpito un nemico, assegnare il punteggio per ogni nemico abbattuto, rilevare se l'astronave urta un nemico o gli elementi dello sfondo che devono essere evitati e così via. Alla fine, il programmatore dovrà creare le routine particolari del programma, come per esempio quella che visualizza l'esplosione del nemico abbattuto, l'esplosione dell'astronave del giocatore, lo scorrimento dello sfondo il movimento di ogni sprite. Tutte queste routine particolari vengono invocate dall'algoritmo di strategia secondo le necessità. Quando la leva del joystick viene spostata a destra, per esempio, lo sfondo deve essere spostato a sinistra per simulare il movimento. Per quanto riguarda la contemporaneità di tutti gli eventi la risposta è semplice: non sono contemporanei. Infatti il computer passa in rassegna tutti gli elementi del videogame e li gestisce a rotazione molto velocemente. Tutto avviene così velocemente che voi non potete accorgervene.

LA DIFESA AD OLTRANZA

...QUANDO TRA IL DISCHETTO E LA SUA UNITÀ DISCO NON ESISTE PIÙ RAPPORTO

Le norme Ansi/Ecma stabiliscono che il dischetto debba avere un "clipping level" non inferiore al 40%. Ciò permette al dischetto di colloquiare con il drive in condizioni di sicurezza.
Ma cosa succede se il drive esce dalle sue norme di sicurezza?
Per esempio se si stara o degrada per invecchiamento o opera in condizioni ambientali critiche?
Solamente un dischetto di caratteristiche superiori, con un "clipping level" del 65%, può continuare il rapporto con la sua unità a disco.
Ecco perchè il SIXTY FIVE difende ad oltranza i tuoi dati.

Sixty Five
Two Sides Double Density
Soft Sectored 48 TPI
REORDER CODE 5501SSF
odp OFFICE DATA PRODUCTS

è un prodotto

VIA AGORDAT, 34
20127 MILANO
Tel. (02) 2871131 (8 linee r.a.)
Telex 315377 SADATA I

VIA CITTÀ DI CASCIA, 29
00191 ROMA
Tel. (06) 3273581 (3 linee r.a.)
FAX (06) 3283894

C.SO MONCALIERI, 259/E
10133 TORINO
Tel. (011) 6967171 (3 linee r.a.)
FAX (011) 6967006